Ramón Invarato Menéndez

Android 100%

Versión 1

100% Compatible con Android 5.0 Lollipop

100% Desde cero a nivel experto

100% Desde cero a nivel experto

100% Patrones

100% Fácil

100% Jarroba

100% Ejemplos completos

100% Respuesta a todas tus preguntas



Con la colaboración de:
Ricardo Moya García
Jesús Alberto Casero Gutiérrez

Índice general

Prólogo

Antes de empezar con este proyecto reflexioné acerca de la manera de obtener el conocimiento desde los inicios de la escritura hasta nuestra manera actual de enfocar el aprendizaje. Me hizo pensar en cómo enfocábamos las cosas, la poca motivación que había al estudiar tecnicismos, y que el método tradicional estaba rotundamente desfasado. No soy un experto en psicología, pero todos somos buenos conocedores de todo cuanto nos ocurre y tenemos perspectiva. Pronto me di cuenta que lo que más me interesaba, me motivaba, me apasionaba, era aquello que podía entender desde la base y podía llevar al mundo real. Tenía que ser ameno, bien explicado, con imágenes cuando fuera necesario, con palabras cuando las imágenes quedaran cortas. Al ser posible siempre ayudado por alguien que ya hubiera tenido la osadía de entrar en ese campo experto.

Recordé tiempo atrás mis años universitarios. Una de las asignaturas que más me incitó a lo que me dedico hoy día fue una humilde optativa, que pudiera haber pasado desapercibida como muchas otras por tener nombres poco agraciados o un poco publicidad. La asignatura se llamada “Computación ubicua”, un nombre que empieza por “computación” cosa normal si se estudia una ingeniería informática; pero lo de “ubicuo” sonaba a cualquier cosa menos a computación en cualquier lugar, donde sea, cuando sea. Cuando me matriculé en esa asignatura ni si quiera tenía un Smartphone, pero fue el momento clave en el que aprendí un montón sobre iOS. Sí, empecé con iOS antes que con Android, daría que pensar que ahora sería experto de iOS; a pesar de ello el destino no lo quiso así. Una serie de circunstancias, como que mi primer Smartphone fuera un Android, me llevaron a estudiar en profundidad todo sobre el sistema operativo del androide verde. Tanto que llevo trabajando con Android desde la versión 1.6.

Muchos años de Android y muchos más de Java me encaminaron hasta esta publicación. He querido recoger el conocimiento de todos estos años y plasmarlo en una obra relacionada con Android, estudiando en profundidad tanto Java como XML. Realmente me parece una de las mejores maneras de aprender a programar para los que están iniciando en este mundo; y no solo de empezar, sino de adquirir conocimientos avanzados de la programación orientada a objetos, programación orientada a eventos, patrones, entre otros muchos que te convertirán en un buen programador.

Tanto si estás leyendo esta obra como desarrollador pasional como programador empresarial, animarte a crear tus propias aplicaciones y subirlas a Google Play. Supongo que estarás conmigo en que pocas satisfacciones hay más que tu propia creación y sobre todo que dé frutos.

Modo de empleo



El libro de "Android 100%" se ofrece completamente **gratuito y actualizado** desde http://jarroba.com/libro-android-100-gratis/

El libro de "Android 100%" está diseñado para obtener una buena base, y convertirse en un experto en el dominio de Android y de paso de Java. Cada tema es incremental al anterior. El libro está estructurado a modo de preguntas muy comunes y sus respuesta, a cada cual dará lugar a nuevas preguntas que serán respondidas en las siguientes.

Cada uno los temas mostrarán **teoría muy detallada**, **cientos de imágenes**, todo en **español**, con fragmentos de **código** que ayudarán a entender los ejemplos completos y **referencias** para completar todavía más el proceso de aprendizaje. **Ejemplos completos de aplicaciones** funcionales y con la **posibilidad de descargar los proyectos**; proporcionaremos todo el **material** necesario para seguir este libro. Todo un **acabado profesional** tanto a nivel de profundidad de las explicaciones, así como la estructura del libro en sí.

Y por supuesto, podrás **hacernos preguntas** en cualquier momento, aportarnos sugerencias, peticiones, seguir a **la comunidad de desarrolladores** en muchos temas de informática técnica, y la posibilidad de aprender del ayudando a otros que están pasando por una situación que ya superaste, **sin ningún coste** en nuestro foro: http://jarroba.com/foro/

Te ofrecemos la oportunidad de conseguir **maestría.** De conseguir uno de los muchos puestos de **trabajo** que están esperándote en el mundo informático. De obtener **reconocimiento** por la creación de aplicaciones de **calidad.** Si compartes la visión por un **mundo tecnológico mejor**, estoy seguro que sabrás agradecer lo que este libro y la página de www.Jarroba.com te harán crecer.

Tú decides cuánto estás dispuesto a aprender, cuánto estás dispuesto a superarte, cuánto quieres dominar sobre tecnología, te ayudaremos en todo cuanto podamos. Todo esto y mucho más en www.Jarroba.com

Si te adoras tanto como nosotros la informática técnica, te gustará seguirnos en:

	Youtube	Jarroba Web https://www.youtube.com/user/Jarrobaweb
	Twitter	@JarrobaWeb https://twitter.com/JarrobaWeb
	Facebook	Jarroba.com https://www.facebook.com/jarrobaWeb
	Google Plus	Jarroba Web https://plus.google.com/+JarrobaWebProfesional/posts
	GitHub	Jarroba Web https://github.com/jarroba?tab=repositories

Apóyanos

Estamos comprometidos al 100% con cada situación personal. Creemos ciegamente en las personas. Todo el mundo es consciente del gran trabajo que supone una obra como ésta, y sabrá agradecer la ayuda que le ha prestado.

Puedes apoyarnos como más lo veas conveniente y justo. Por ejemplo, compartiendo nuestro trabajo o realizándonos una donación. Eres completamente libre de elegir la gratitud. Considera que este trabajo lo has hecho tú mismo, valora qué hubieras deseado recibir por este trabajo y cuánto te gustaría su continuidad.



Ponemos a disposición de todo el mundo un apartado de donaciones en:

https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=2QT2VY3APMF3Q

Introducción

¿Qué se puede hacer con Android?



Algunos **ejemplos** son:

- Navegar y buscar por el mundo
- Conectarse y compartir
- Entretenimiento digital
- Crear y colaborar



¿Qué tiene Android?





Un **resumen** es:

- App Widgets para el escritorio
- Notificaciones
- Multi-tarea
- Reconocimiento de voz
- Cámara de fotos y vídeos

¿Un resumen de la historia de Android?

Desarrollado por **Android Inc desde 2003.** Hasta que fue **comprado por Google en 2005**.

Se liberó el **código bajo licencia Apache** al crearse la Open Handset Alliance el 5 de Noviembre de **2007.** También considerado en cumpleaños de Android.

En **2008** se crean los primero chips compatibles y se lanza el **primer teléfono Android**, el HTC Dream.

Se empiezan a nombrar como **dulces a las versiones** de Android a partir de **2009.**

¿Qué características suelen tener los dispositivos con Android?

Gráficos: VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0

Almacenamiento: SQLite

Conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WiMAX

Mensajería: SMS, MMS y C2DM

Navegador Web: WebKit, motor JavaScript V8

Multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP

Streaming: RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5 <video> tag), Adobe Flash Streaming (RTMP)

Hadware: cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.

Bluetooth: A2DF y AVRCP, el envío de archivos (OPP)

Videollamadas, Voz, Multi-táctil

Tethering: usar al dispositivo como punto de acceso inalámbrico

Referencias:

- http://es.wikipedia.org/wiki/Android
- http://visual.ly/sweet-history-android
- http://es.wikipedia.org/wiki/Android

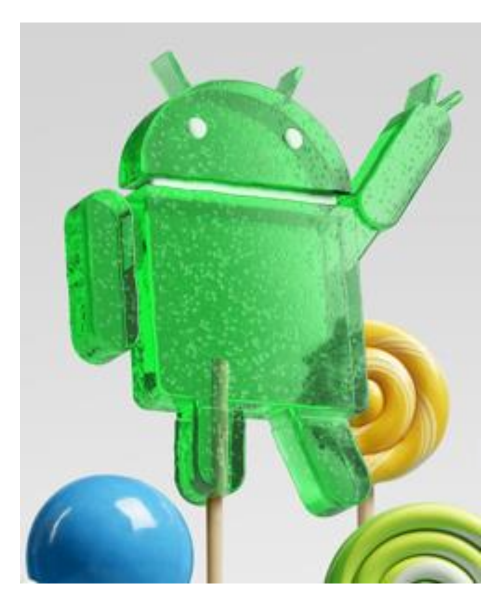
Estado actual de Android

¿Qué versiones existen de Android?

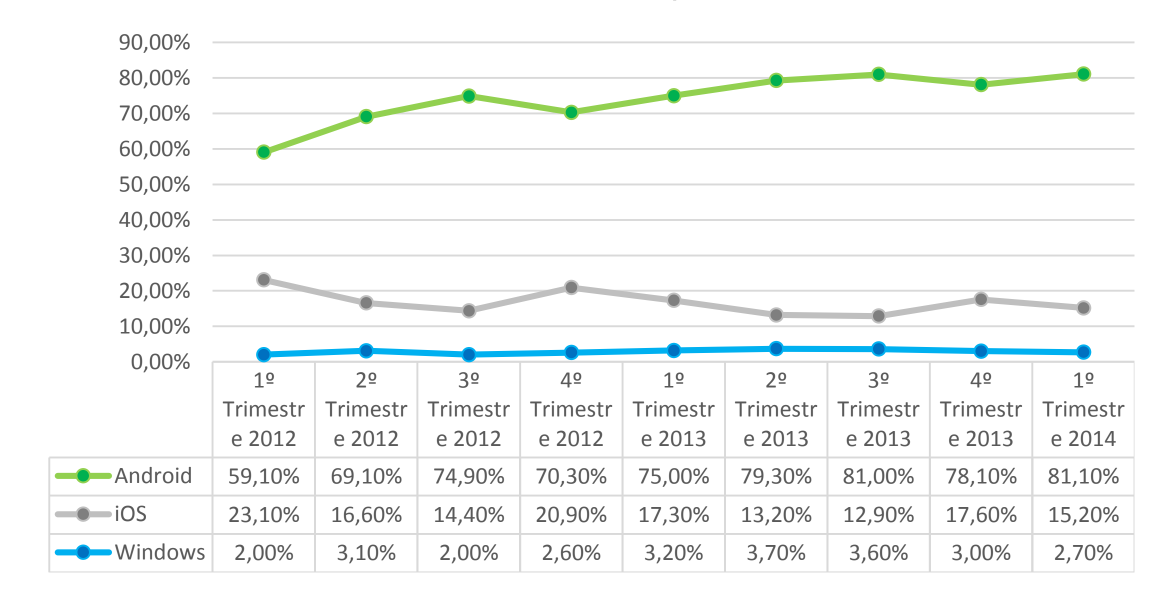
Un sistema operativo Android tiene tres tipos de denominar a las versiones de Android, aunque las tres hacen referencia a la misma versión:

- La **comercial** con el nombre de postre. Por ejemplo: KitKat
- La de los **fabricantes** (y también comercial) con la versión y subversión. Por ejemplo: 4.4
- La de **desarrollador** con el nivel del API (ésta nos interesa mucho para desarrollar en Android): Por ejemplo: 19

Nombre	Versión	API
Beta	0	
Apple Pie	1.0	
Banana Bread	1.1	
Cupcake	1.5	3
Donut	1.6	4
Eclair	2.0/2.1	7
Froyo	2.2	8
Gingerbread	2.3.2/2.3.7	9/10
Honeycomb	3.0/3.1/3.2	11/12/13
Ice Cream Sandwich	4.0/4.0.3	14/15
Jelly Bean	4.1.2/4.2.2/4.3	16/17/18
KitKat	4.4/4.4(Wear)	19/20
Lollipop	5.0	21

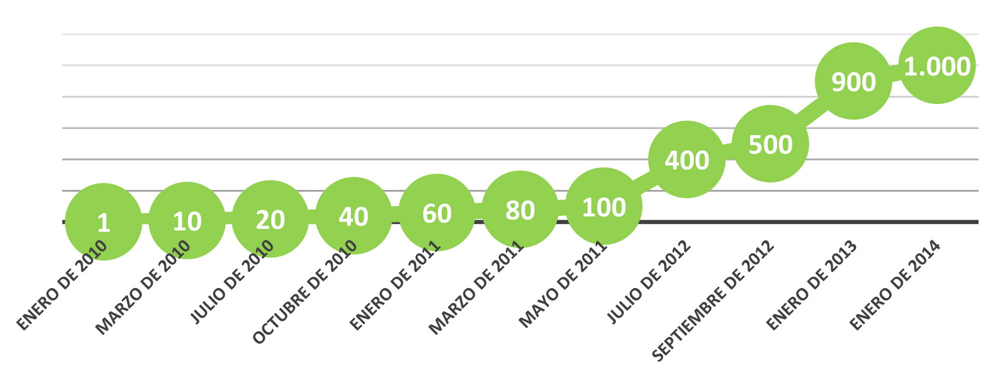


¿Cuál es la cuota de mercado a nivel mundial de los Sistemas Operativos móviles?



	1º Trimestre 2012	2º Trimestre 2012	3º Trimestre 2012	4º Trimestre 2012	1º Trimestre 2013	2º Trimestre 2013	3º Trimestre 2013	4º Trimestre 2013	1º Trimestre 2014
Android	59,10%	69,10%	74,90%	70,30%	75,00%	79,30%	81,00%	78,10%	81,10%
iOS	23,10%	16,60%	14,40%	20,90%	17,30%	13,20%	12,90%	17,60%	15,20%
Windows	2,00%	3,10%	2,00%	2,60%	3,20%	3,70%	3,60%	3,00%	2,70%

¿Cuántos dispositivos activos hay de Android?



Se estima que en 2014 se activen unos 1,5 Millones de dispositivos cada día

¿Cuál es la cuota de mercado a nivel mundial de las diferentes versiones de Android?

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	11.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	9.6%
4.1.x	Jelly Bean	16	25.1%
4.2.x		17	20.7%
4.3		18	8.0%
4.4	KitKat	19	24.5%

Ilustración 1 - Imagen obtenida de la web de http://developer.android.com/. Datos recogidos durante un periodo de 7 días al finalizar el 9 de Septiembre de 2014 (Todas las versiones con una distribución de menos de un 0,1% no se mostrarán)

¿En qué versión es aconsejable desarrollar en el año 2014?

El autor de este libro recomienda a nivel general (salvo casos de necesitar acaparar cerca del 100% del mercado) **centrarse en el desarrollo del número más alto de la versión mayor** (a día de hoy 5.x). Motivos:

- Las versiones previas tienden a desaparecer o a ser actualizadas
- Las versiones antiguas no soportan muchas de las nuevas características (si las soportan con límites, como ActionBar), presentan problemas de seguridad, de rendimiento y están menos optimizadas a nivel energético

De necesitar **más cuota de mercado**, la relación de mayor cuota de mercado con la versión mayor que más características soporta se obtiene al desarrollar desde la **versión 4.0.**

Versiones que no hay que tener en cuenta, debido a que apenas existan:

- **Versión 1.x.** Demasiado antigua. La versión más moderna fue la 1.6, fue lanzada en Septiembre de 2009
- **Versión 3.x.** Era solo para Tablets, casi todas están actualizados a la 4.x (Casi todos los dispositivos que se vendieron con la versión 3.X pueden, en potencia, actualizarse a la versión 4.X, si no está actualizado es por falta de interés del poseedor del dispositivo)

Referencias:

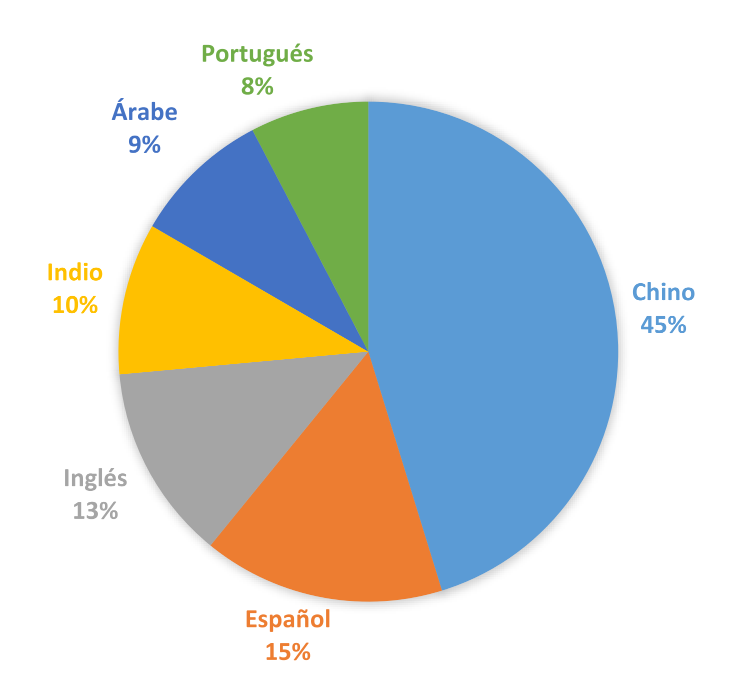
- http://developer.android.com/about/dashboards/index.html
- http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android
- http://es.engadget.com/2013/10/31/android-cuota-mercado-81-por-ciento/
- http://www.idc.com/getdoc.jsp?containerId=prUS24108913
- http://www.idc.com/getdoc.jsp?containerId=prUS24257413
- http://www.idc.com/getdoc.jsp?containerId=prUS24442013
- http://www.idc.com/getdoc.jsp?containerId=prUS24676414
- http://www.idc.com/prodserv/smartphone-os-market-share.jsp
- http://www.youtube.com/watch?v=1CVbQttKUIk
- http://www.idc.com/getdoc.jsp?containerId=prUS24857114
- http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/
- http://www.androidpit.com/schmidt-android-activations-1-5-million-a-day-1-billion-by-2014

Idiomas

¿Cuáles son los idiomas más hablados en el mundo?

Hemos escogido seis más hablados, de los cuales, casi todas las personas sobre la Tierra conocen al menos uno si no es por ser primer idioma por segundo (sé que esto es debatible, pero bueno, pongo mi opinión, así puedes trabajar con dos opiniones, la tuya y la mía ☺):

Posición	Idioma	Hablantes en millones
1º	Chino	1.197
2º	Español	414
3º	Inglés	335
4º	Indio	260
5º	Árabe	237
6º	Portugués	203



¿En qué idiomas se aconseja traducir nuestra aplicación?

Evidentemente una buena opción para elegir será la mayor relación de hablantes o personas que entiendan un idioma, y la posibilidad de que alguien se descargue o compre nuestra aplicación.

Por ello el **inglés** debería de ser la opción más aconsejable, y el idioma por defecto para las carpetas de recursos (como veremos en el tema de Recursos). Además, si estás leyendo este libro no creo que tengas ningún problema en hacer la aplicación en **español**. Aunque con estos dos idiomas cubriríamos un 28% de hablantes, los que al menos entienden alguno de estos dos son bastante mayor que cualquier otro idioma.

Otros idiomas importantes son el ruso y el chino.

Como países emergentes en los que se habla los siguientes idiomas, es importante el indio, el árabe y el portugués. Tienen mucha cuota de mercado y son muy aficionados a descargar aplicaciones, sobre todo los de habla portuguesa (por experiencia Brasil descarga muchas Apps).

Referencias:

- http://www.ethnologue.com/statistics/size
- http://jakubmarian.com/map-of-the-percentage-of-people-speaking-english-in-the-eu-by-country/
- http://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s
- http://es.wikipedia.org/wiki/Idioma_espa%C3%B1ol

Arquitectura

Aplicaciones: cualquier tipo de aplicación escrita en Java.

Framework de las aplicaciones: Acceso al API para rehusar componentes o modificarlos.

Bibliotecas en C/C++: el desarrollador puede usarlas a través del Framework.

Runtime de Android: bibliotecas del lenguaje Java y única instancia en la máquina virtual Dalvik.

Núcleo Linux: Capa de abstracción del hardware y servicios de seguridad, gestión de memoria, de procesos, pila de red, modelo de los controladores, etc.

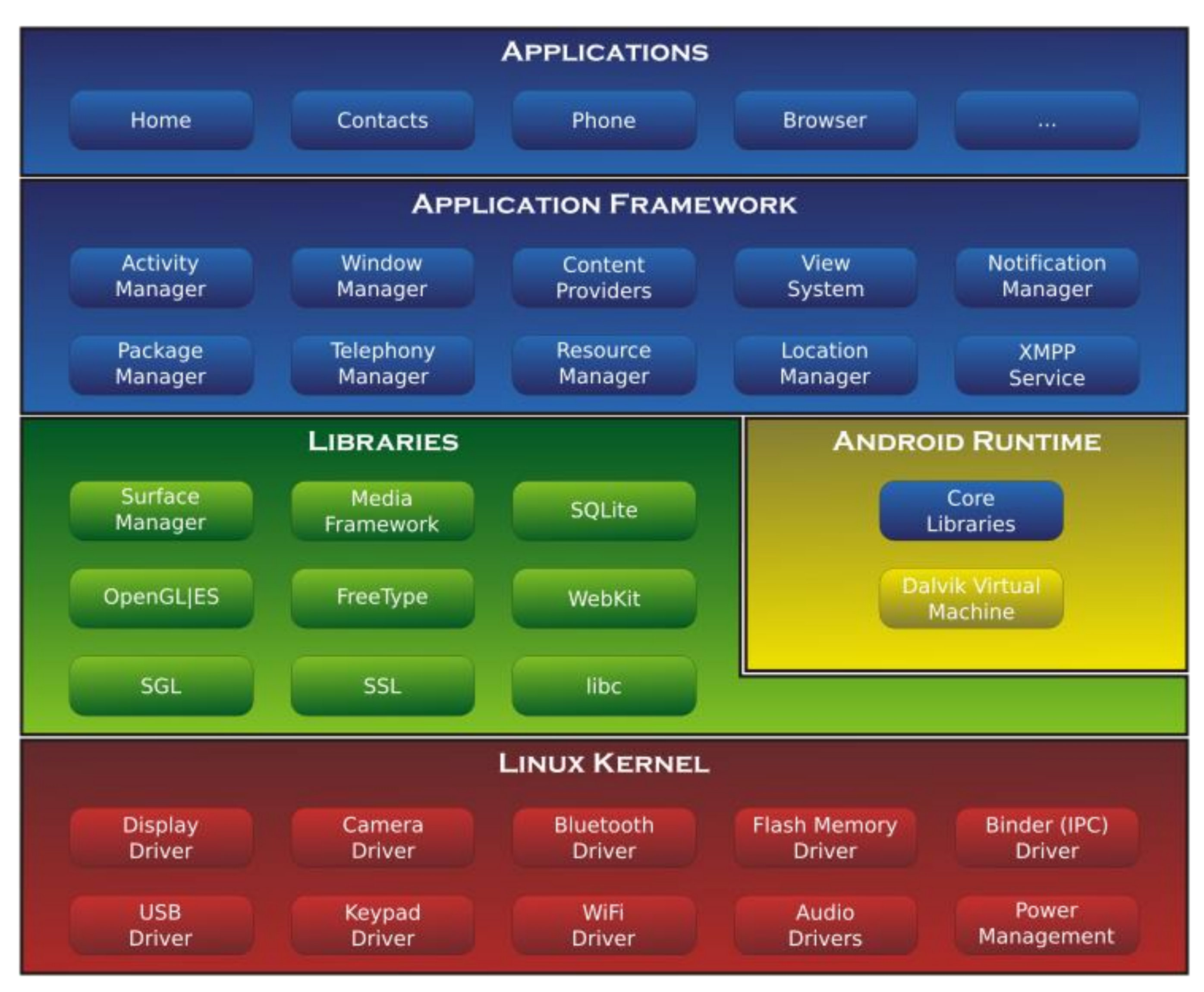


Ilustración 2 - Imagen de la arquitectura de Android obtenida de la Wikipedia

Referencias:

- http://en.wikipedia.org/wiki/Android_(operating_system)

JDK (Java Development Kit)

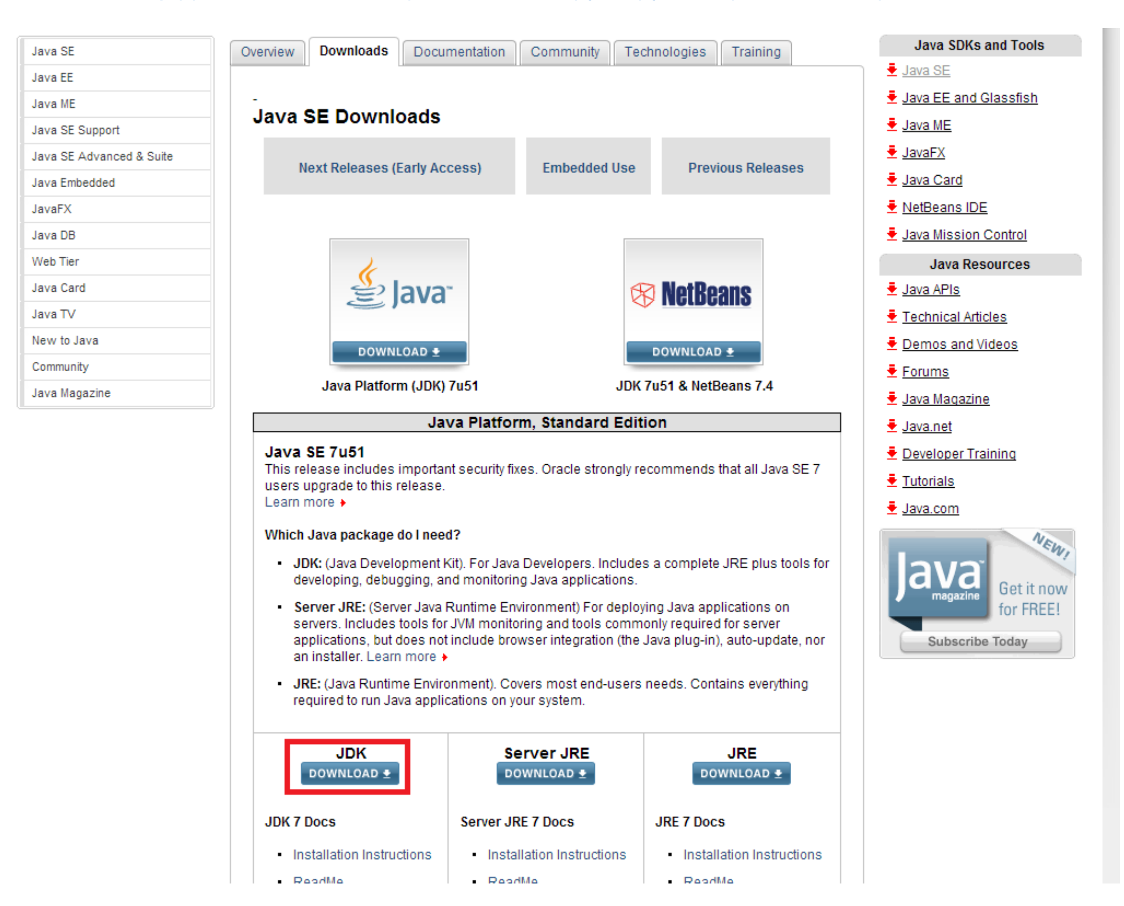


¿Qué contiene?

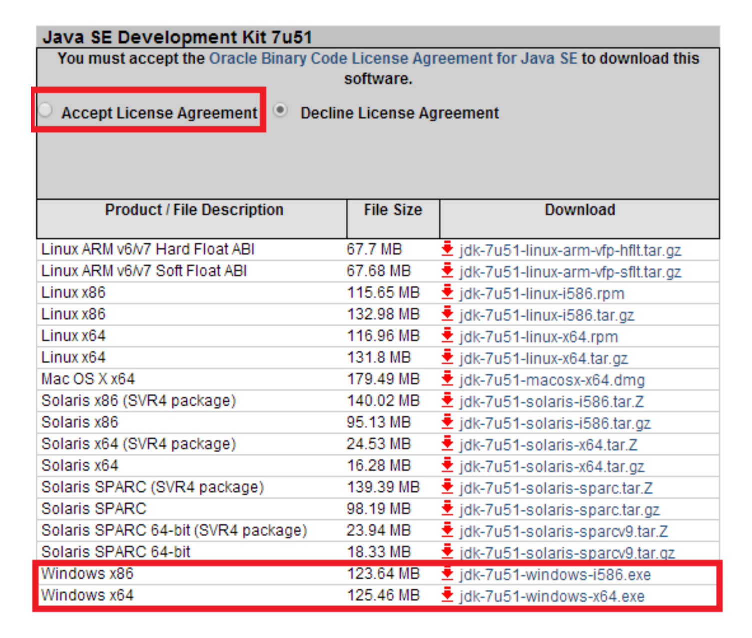
- JRE (Java Runtime Environment)
- Herramientas para el desarrollo
- Herramientas para la depuración
- Herramientas para la monitorización de las aplicaciones Java

¿Cómo conseguirlo e instalarlo?

1. Elije JDK en: http://www.oracle.com/technetwork/java/javase/downloads/index.html



2. Aceptamos la licencia y escogemos el JDK de nuestro sistema operativo y de la arquitectura (si no sabes que arquitectura es, de x86 o de x32, lee la nota a continuación). Comenzará la descarga de unos 130 Mb



Java SE Development Kit 7u51

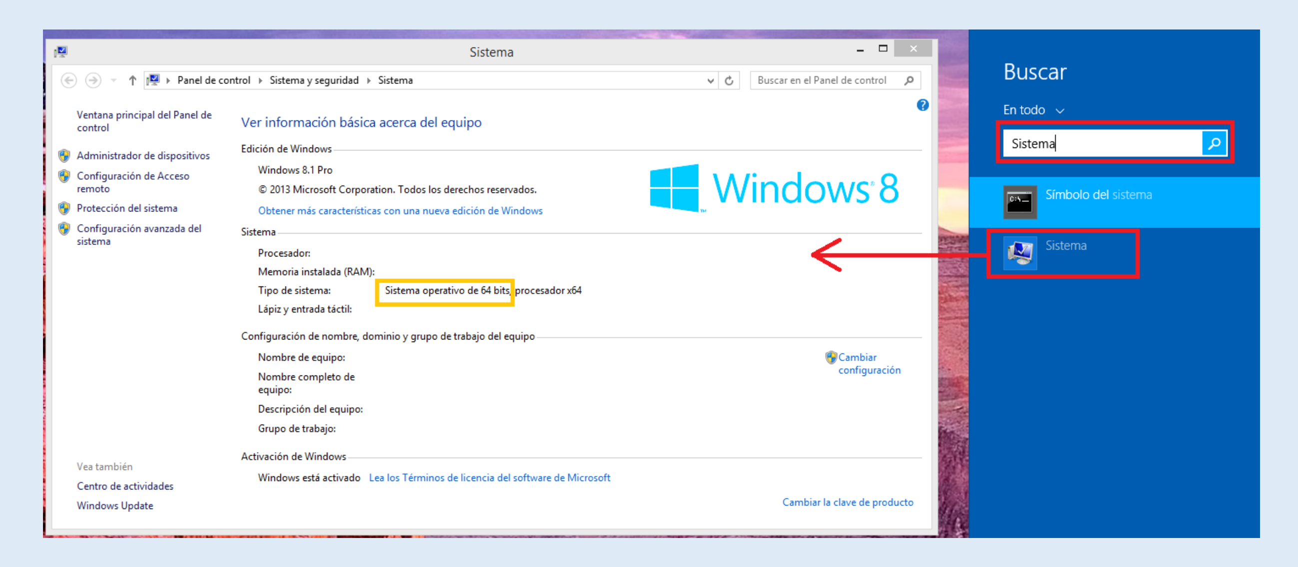
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement   Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

¿Cuál es la arquitectura de mi sistema operativo Windows?

En Desde Windows Vista en buscar escribe “sistema”, y elige “Sistema”. En la ventana que se abre en “Tipo de sistema” aparecen la arquitectura del sistema operativo



Cualquier otra duda al respecto puedes consultarlo en la página oficial en: http://windows.microsoft.com/es-es/windows/32-bit-and-64-bit-windows#1TC=windows-7

3. Instalamos el JDK que se ha descargado.

Eclipse + ADT (Entorno de desarrollo Integrado)

¿Qué contiene este paquete?



- Editor de código
- Compilador
- Depurador
- Gestor de pruebas unitarias
- Constructor de interfaces gráficas
- Gestor de plugins
- Plugin ADT (Android Developer Tools)
- Android SDK Tools

¿Qué es el ADT y qué contiene?

El ADT (Android Development Tools) es un Plugin para Eclipse viene con:

- Creación de proyectos
- Gestor de empaquetado e instalación
- Depurador de errores
- Integración con las herramientas del SDK
- Documentación para las APIs framework de Android
- Recursos enlazados mediante el fichero R.java
- Editores de programación Java, editores de XML y editores gráficos de diseños de pantallas
- Ayudas para la refactorización de los recursos
- Gestor de actualizaciones del propio ADT

¿Qué contiene el SDK?

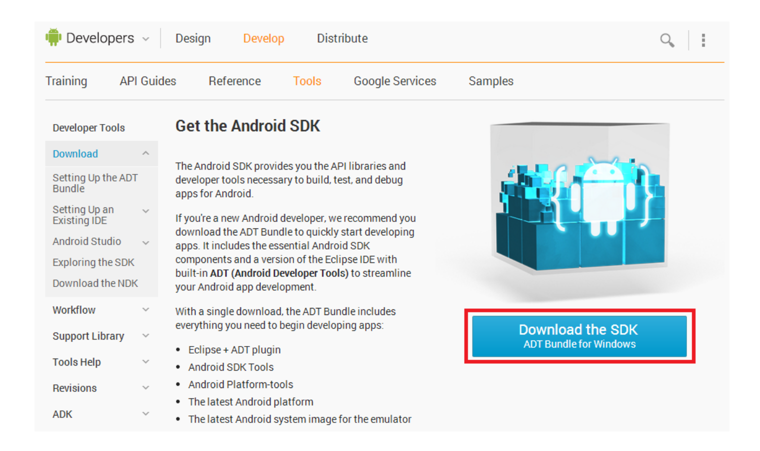
El SDK (Software Development Kit) es un paquete que incluye:



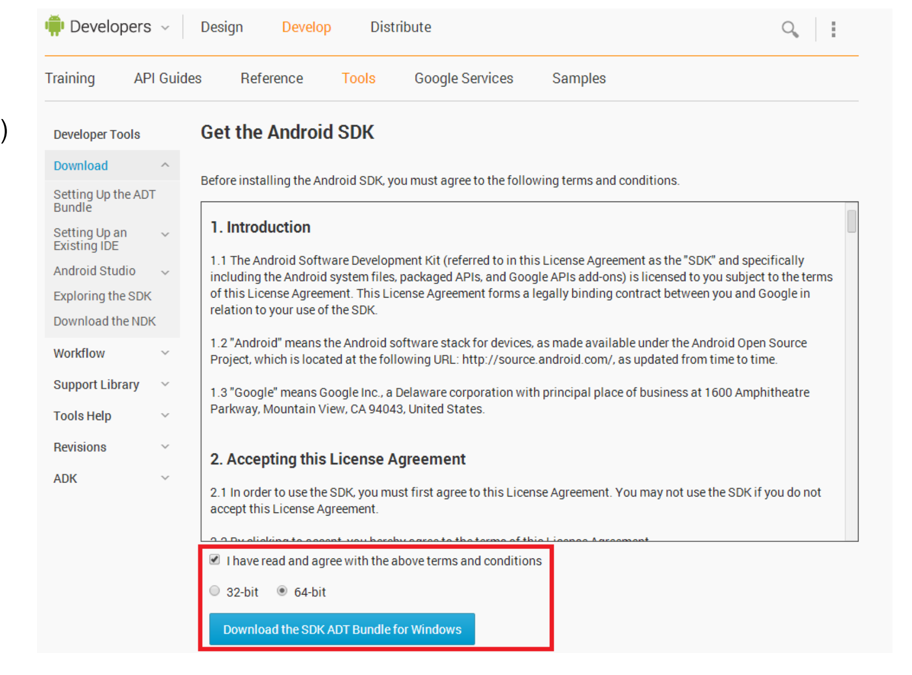
- Depurador
- Biblioteca Android
- Simulador (ADB)
- Documentación
- Ejemplos de código
- Tutoriales

¿Cómo conseguirlo e instalarlo?

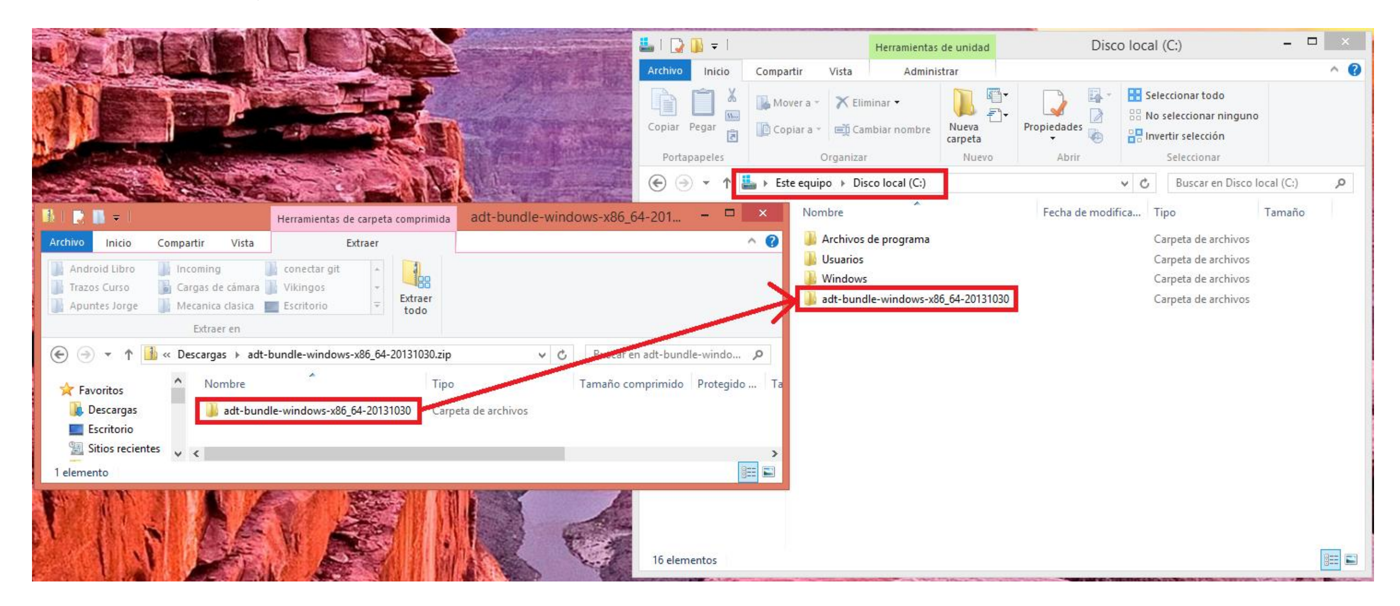
1. Pulsa el botón de descargar de: http://developer.android.com/sdk/index.html



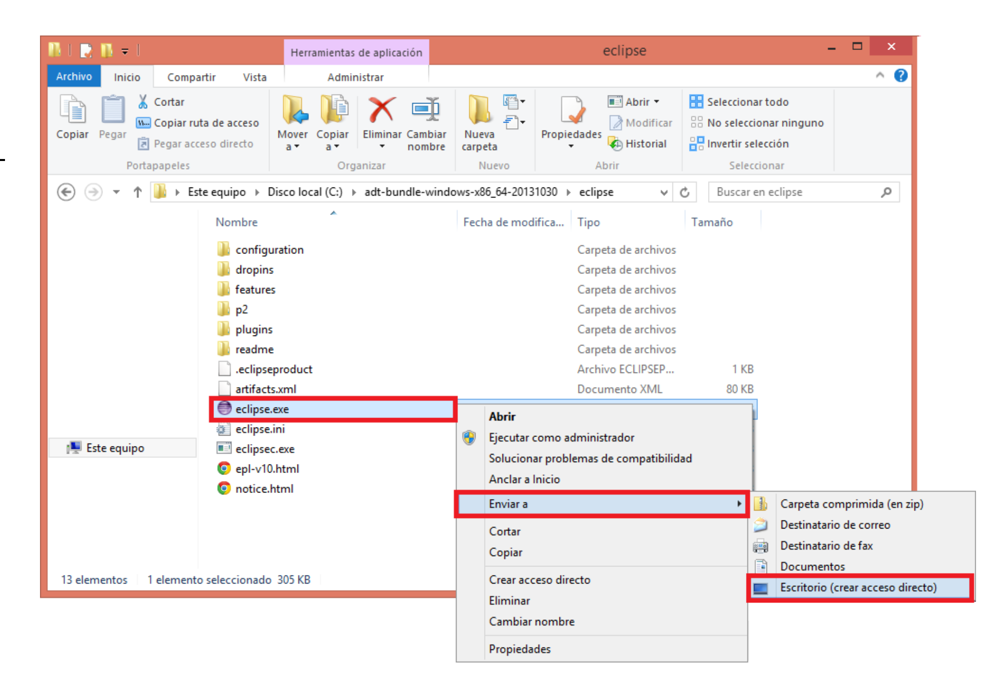
2. Acepta el contrato, elige la arquitectura de tu sistema operativo (Si no sabes cuál poner, lee la nota a continuación) y pulsa en descargar para que comience la descarga de unos 500Mb



3. Descomprimimos el fichero que se ha descargado en algún sitio del ordenador donde no lo vayamos a mover. Por ejemplo directamente en la unidad C.

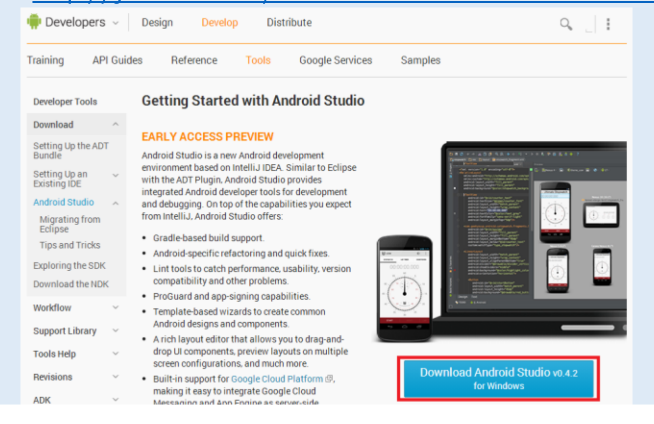


4. Enviamos al escritorio el acceso directo de Eclipse que se encuentra en el ejemplo en la ruta “C:\adt-bundle-windows-x86_64-20131030\eclipse”



¿Y Android Studio?

Tanto Android Studio como Eclipse son semejantes, puedes elegir cualquiera. En este libro hemos decidido hablar Eclipse para empezar el desarrollo con Android debido a que hay mucha más comunidad para Eclipse recomendamos empezar por este y no por Android Studio, ya que todavía está en beta.
De cualquier manera, para seguir este libro sobre Android puedes hacerlo con cualquier entorno de desarrollo.
Puedes ver como instalarlo en: http://jarroba.com/introduccion-a-android-studio-video/



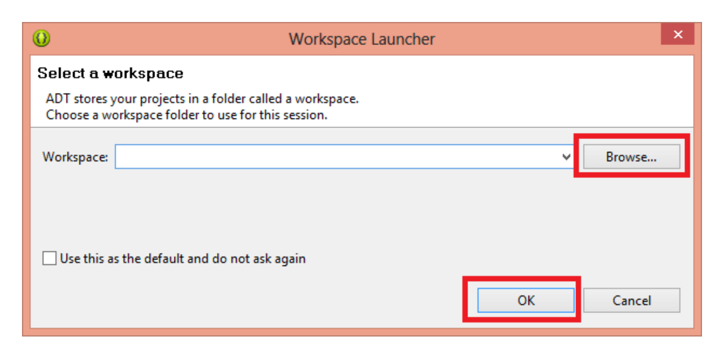
Referencias:

- http://jarroba.com/aprender-a-programar-conociendo-lo-que-es-un-entorno-de-desarrollo-integrado-ide/
- http://www.eclipse.org/
- http://developer.android.com/sdk/index.html
- http://developer.android.com/tools/help/adt.html
- http://developer.android.com/sdk/index.html

Nuestro espacio de trabajo

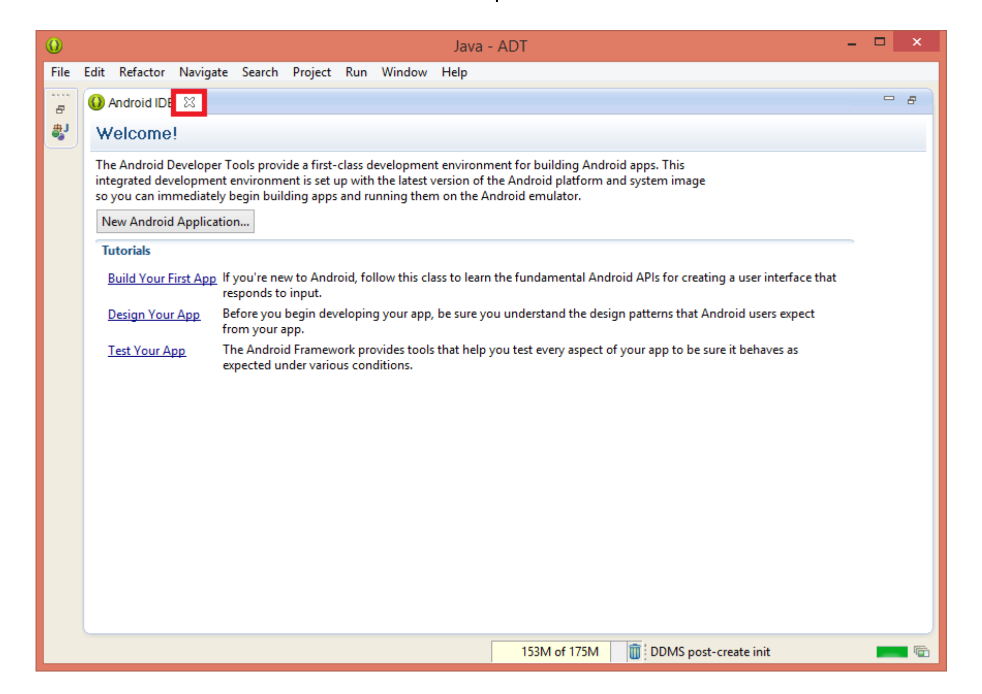
¿Cómo inicio mi espacio de trabajo?

1. Iniciamos Eclipse desde el acceso directo que creamos previamente en el escritorio, o desde el ejecutable "eclipse.exe"

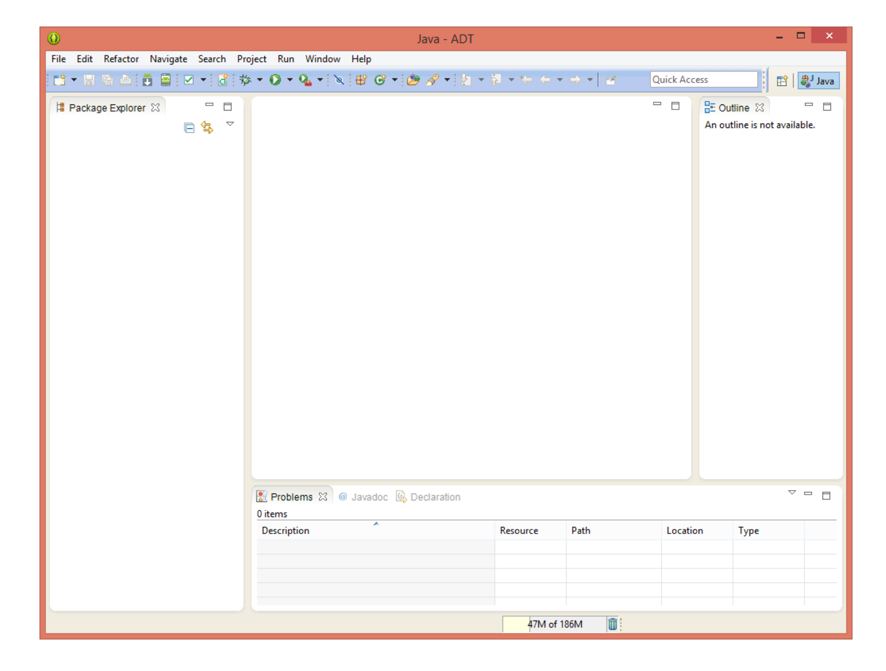


2. En la ventana que se nos abre elegiremos una carpeta de nuestro ordenador donde colocar el espacio de trabajo (Creamos una carpeta vacía donde queramos y la elegimos en esta ventana)

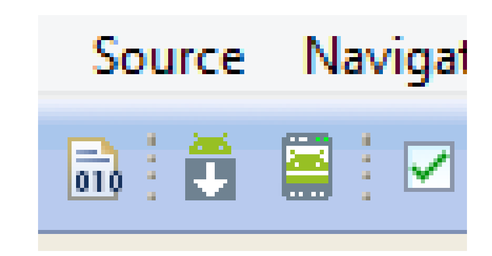
3. Se abrirá Eclipse. Lo primero que nos mostrará será la pantalla de bienvenida que cerraremos pulsando en la X



4. Ya estaremos en nuestro entorno de programación.

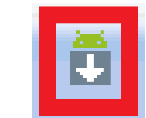


Android en Eclipse y configuración



Nos fijamos en los dos iconos de Android en la barra de tareas (también están en menú "Windows").

SDK Manager

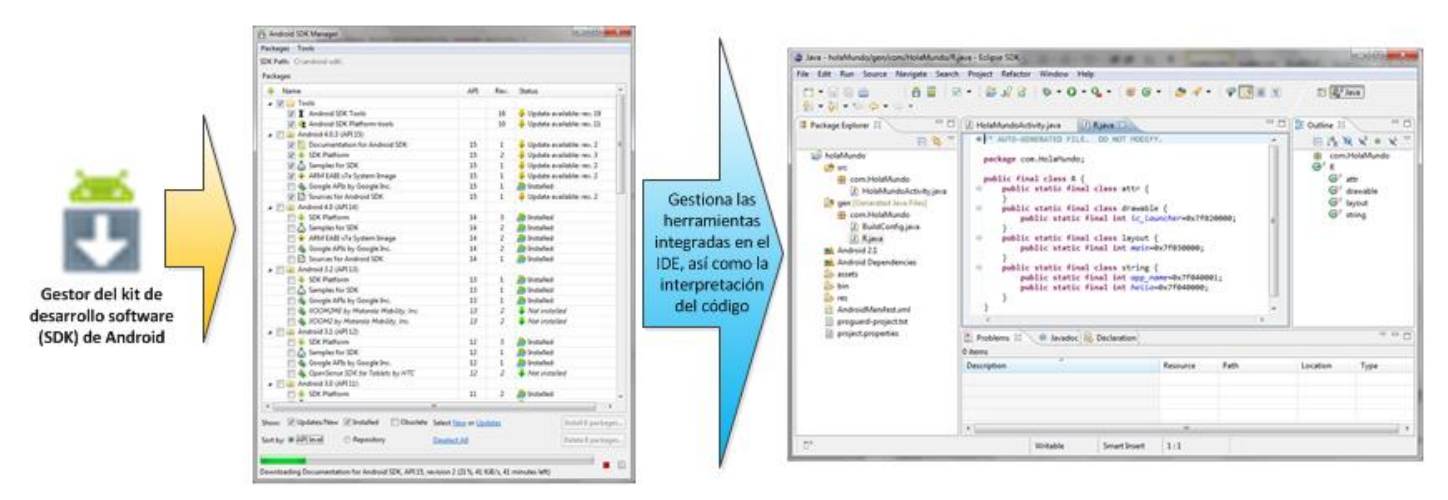


¿Qué significa?

Gestor del kit de desarrollo software de Android

¿Para qué sirve?

Para gestionar los paquetes de las APIs como las versiones de Android para desarrollar, drivers, etc

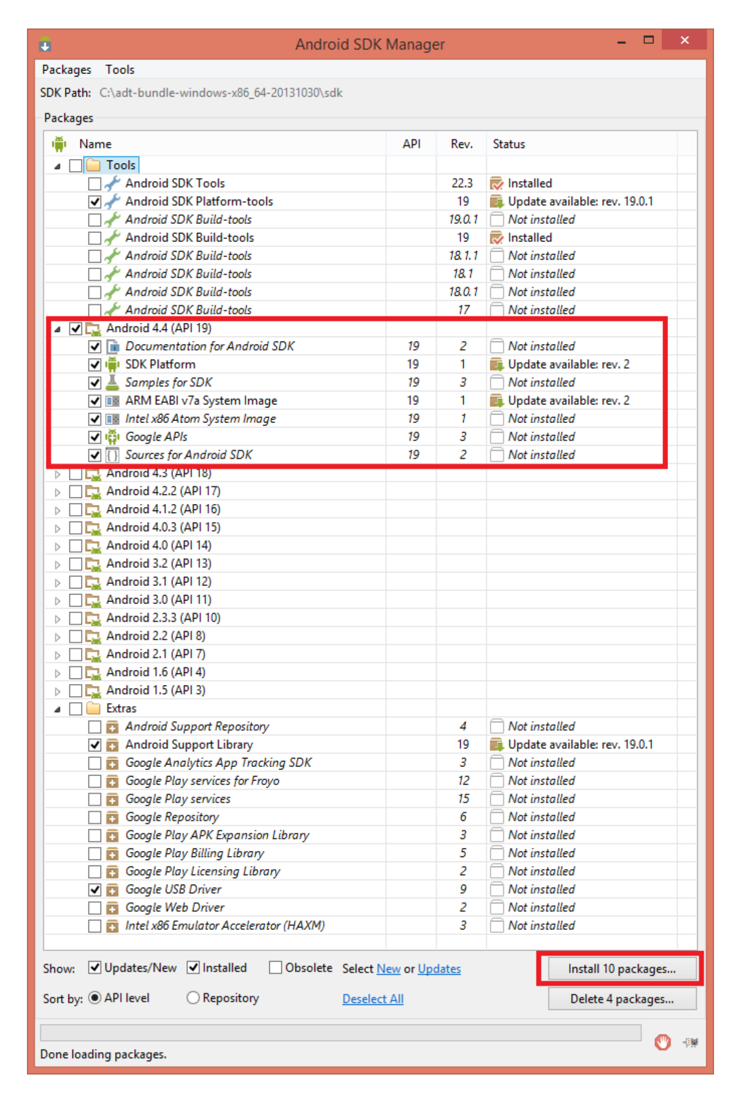


¿Cómo se utiliza?

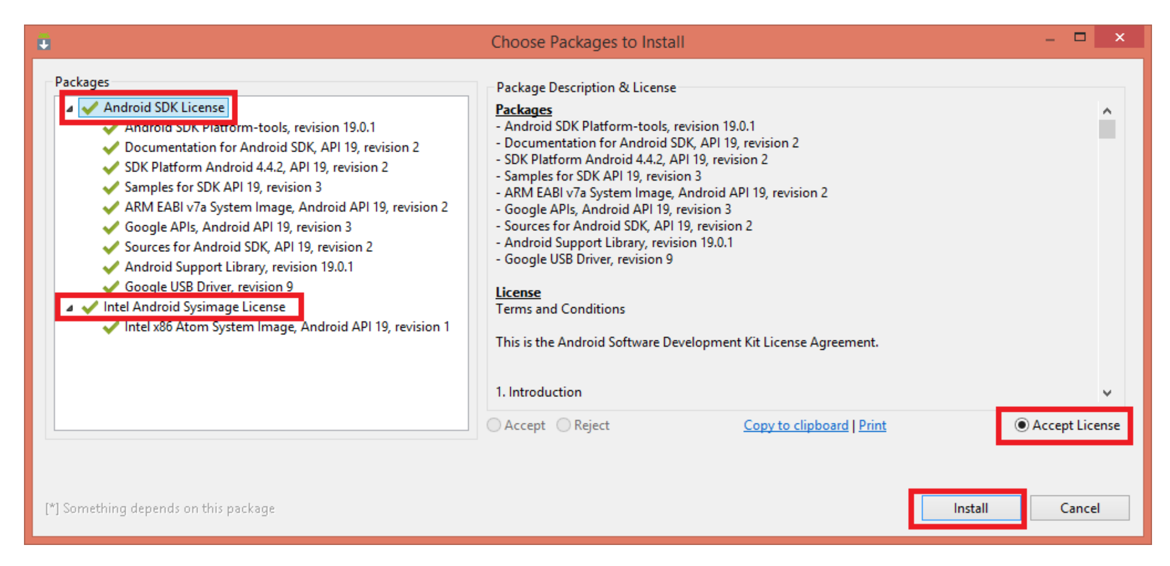
Hacemos clic en su icono y se nos abrirá una ventana en la que esperaremos a que termine la barra de carga inferior.

Aquí podremos marcar otras versiones de los sistemas operativos que queramos emplear principalmente en el emulador de Android (recomiendo primero el más actual que haya, a día de hoy la 4.4.2, y luego si se quieren instalar algunas como la 2.1, 2.3.3 y 4.0). Por defecto se seleccionará para instalar la última versión, de la que recomiendo instalar todo su contenido de su paquete (ya que viene con ejemplos y algunas utilidades de las que sacaremos provecho). También podemos elegir otros paquetes, o incluso seleccionarlo todos (son cerca de 10Gb).

Da igual que elijamos algo que nos pueda interesar o nada más, es necesario que siempre instalamos lo que el SDK Manager nos recomienda pulsando el botón de "Install X packages…".



Al hacerlo nos llevará a una pantalla para aceptar las licencias. Los contratos que son iguales los podremos aceptar todos juntos; los que no tendremos que aceptar también aparte y pulsar el botón “Install”.

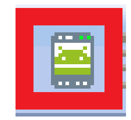


Esperaremos a que descargue todo, podremos entonces cerrar el SDK Manager y volver a Eclipse

Referencias:

- http://developer.android.com/tools/sdk/tools-notes.html

AVD (Android Virtual Device Manager) Manager

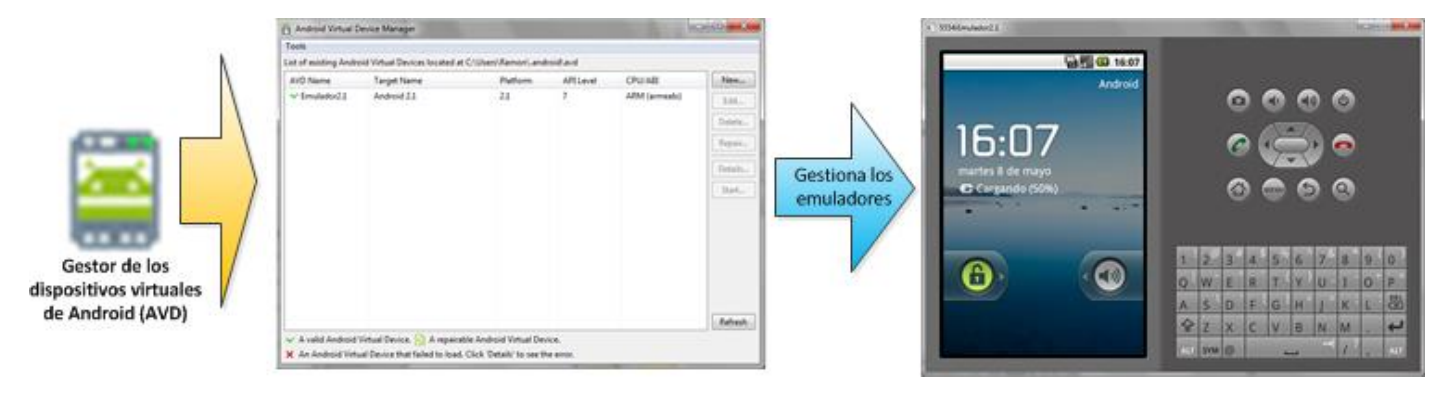


¿Qué significa?

Gestor de los dispositivos virtuales de Android, es decir, los emuladores

¿Para qué sirve?

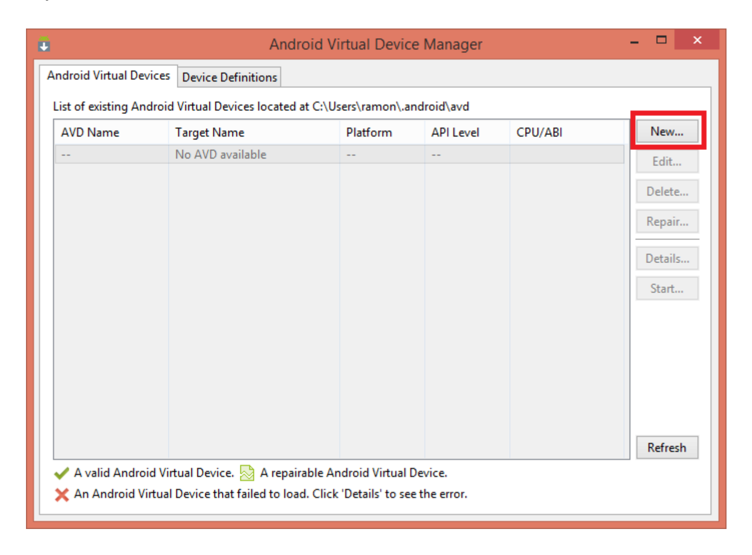
Para gestionar los emuladores de Android, cosas como la versión del sistema operativo que tendrán, el tamaño que ocuparán en pantalla, cuanta memoria consumirán, etc



¿Cómo se utiliza?

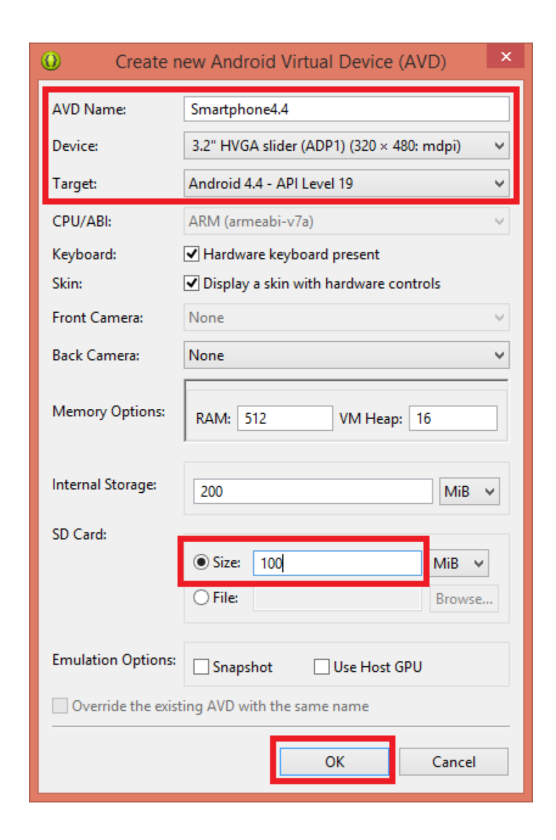
Seleccionamos este icono para que se nos abra la ventana que estará vacía.

Creamos un nuevo emulador pulsando en "New…"

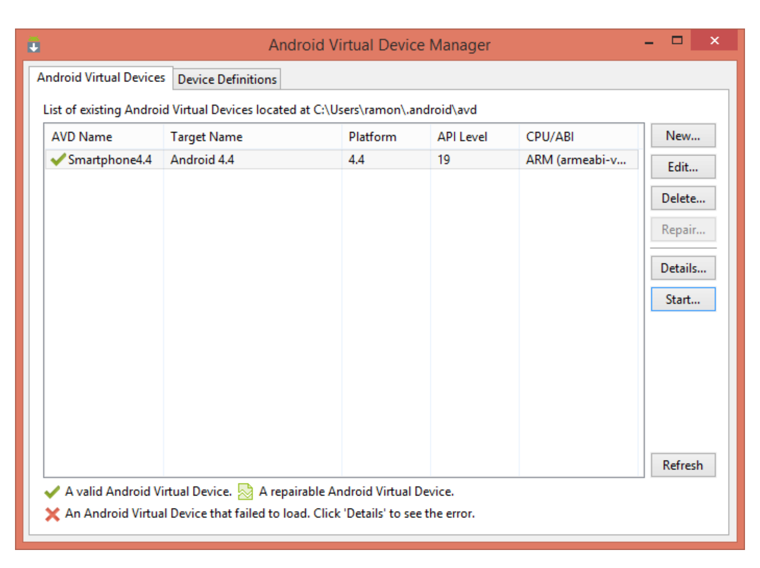


Se nos abrirá otra ventana para configurar el emulador. Aquí definiremos cosas como el nombre del emulador, el tamaño de pantalla y resolución del dispositivo que utilizaremos, la versión del sistema operativo de Android, entre otros.

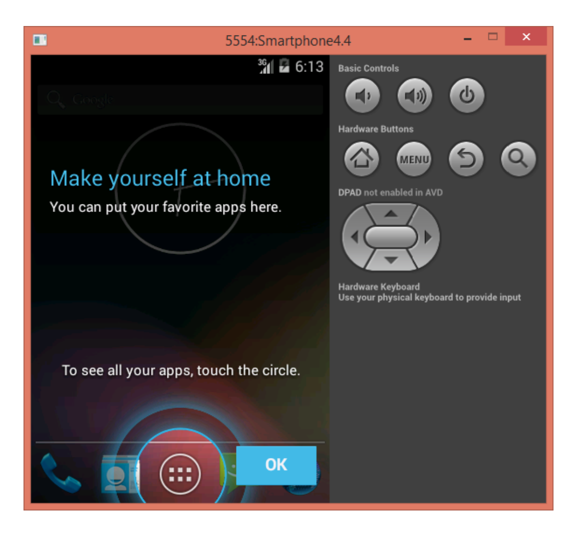
Recomiendo para empezar definir solo "AVD Name" con un nombre identificativo que nos indique el tamaño (como si simulará un Tablet o un Smartphone) y la versión de Android que elegiremos. En "Device" escogeremos un tipo de pantalla de dispositivo; hay muchos, recomiendo usar una de baja resolución, sobre todo si tenemos ordenadores poco potentes y para que el tamaño en pantalla sea manejable (podremos escalar estas pantallas por si no cupieran en nuestra pantalla del ordenador en el que vamos a desarrollar). También un "Target" que será la versión del sistema operativo que se iniciará con nuestro emulador. Recomiendo poner algo de memoria a la "SD Card" pues la necesitaremos más adelante (y para que no nos esté avisando Android todo el rato de que no hay tarjeta de memoria insertada); pero cuidado, todo este tamaño que definamos aquí será memoria de disco duro consumida (definir un tamaño pequeño, ya que se ocupará esa memoria al instante de crear el emulador). Pulsamos "OK" para terminar.



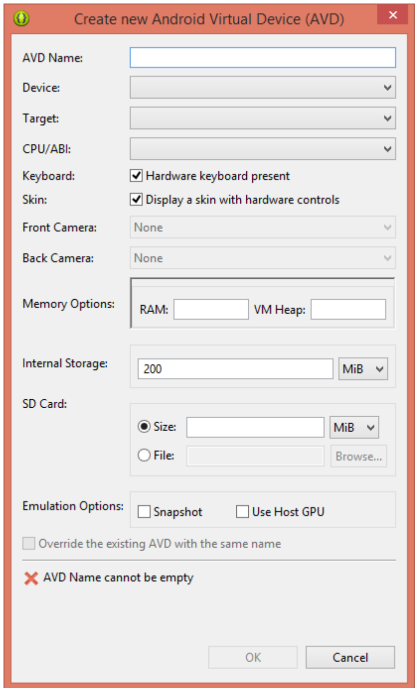
Ya lo tendremos configurado



Podemos iniciar el emulador al pulsar “Start...”. Aunque se puede hacer desde aquí, también desde Eclipse, desde donde es más cómodo para el desarrollo. Podemos cerrar la ventana del AVD y volver a Eclipse.

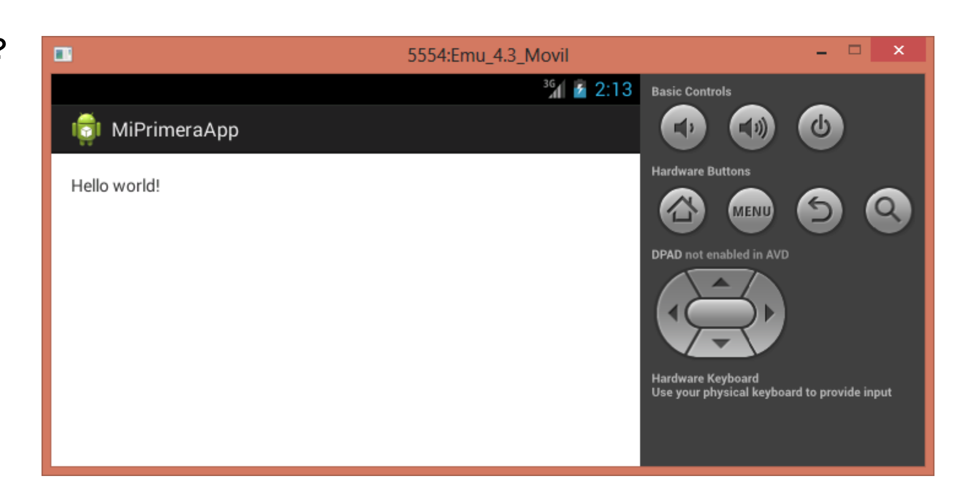


¿Para qué sirve cada configuración del emulador?



- **AVD Name**: ponemos un nombre identificativo (recomendable poner la versión del sistema operativo o el nivel de la API), como por ejemplo "Emu_4.3_Movil"
- **Device**: Elegimos una resolución de pantalla acorde con la potencia de nuestro ordenador (recomiendo la densidad de "HVGA" (Por ejemplo, 3.2" HVGA slider (ADP1) (320 x 480; mdpi)). Por dos razones, una porque no exige un ordenador muy potente al ser de 320×480 hay pocos píxeles que procesar; y segunda, para que sin tocar más, el emulador tenga un tamaño manejable en pantalla para la mayoría de los monitores). Para saber que significa cada sigla podemos ver una tabla con todas en: http://developer.android.com/guide/practices/screens_support.html
- **Target**: Elegimos el sistema operativo que instalaremos en el emulador que estamos fabricando.
- **CPU/ABI**: Aquí se puede elegir que procesador simulado queremos que tenga nuestro emulador. Recomendamos que selecciones "ARM (armeai-v7a)". Seleccionaremos otros para otras arquitecturas o para poder emplear tecnologías de aceleración del emulador de Android.
- **Keyboard**: Si queremos poder usar nuestro teclado físico (el físico del ordenador, el que está junto al ratón) o no (en este caso solo podremos hacer uso del de pantalla). Recomendamos marcar la casilla.
- **Skin**: Para mostrar los botones físicos (como atrás, home, volumen, apagar, etc). Recomendamos marcar la casilla.
- **Front Camera**: Como queremos simular la cámara frontal.
- **Back Camera**: Como queremos simular la cámara trasera.
- **Memory Options**: Definimos la memoria RAM de unos 512 y VM Heap de unos 16 (espacio de almacenamiento dinámico para la máquina virtual; es el espacio necesario para tener una instancia de la máquina virtual Dalvik).
- **Internal Storage**: Para definir la memoria interna de unos 200 MegaBytes (que es lo que está por defecto como "MiB").
- **SD Card**: en el campo "Size" ponemos una cantidad de 100 MegaBytes (que es lo que está por defecto como "MiB"). Ponemos una cantidad no muy elevada, ya que todo lo que pongamos será espacio que el emulador reserve en el disco duro.
- **Snapshot**: De momento la casilla quedará desactivada hasta que veamos más adelante como usarlo. Como en algunos sistemas operativos, permite poner el emulador en suspensión para que cargue en el estado en el que lo dejamos la última vez (ahorra el tiempo de tener que esperar a que el sistema operativo se inicie). Así podremos iniciar el emulador más rápidamente.
- **Use Host GPU**: Para usar la GPU de tu tarjeta gráfica (Usará las librerías de OpenGL instaladas por los drivers gráficos del ordenador). Si no vas a crear videojuegos, mejor déjala desmarcada de momento (más adelante podrás editarlo).

¿El Emulador tiene atajos de teclado?



Teclas físicas del dispositivo:

- **Esc**: Atrás
- **Inicio**: Home
- **F2**: Menú
- **F7**: Bloqueo y apagado
- **+**: Subir volumen
- **-**: Bajar volumen
- **F5**: Buscar
- **F6**: TrackBall

Vista del dispositivo:

- **7**: Cambiar la orientación
- **Alt + Entrar**: Pantalla completa

Referencias:

- http://jarroba.com/preparar-las-herramientas-necesarias-para-programar-en-android/
- http://developer.android.com/tools/devices/index.html

Usar un dispositivo real en vez de un emulador

Un dispositivo móvil es muy recomendable para utilizarlo para ultimar pruebas de una aplicación o directamente para poder probar características que el emulador no soporte (como la detección de más de un dedo en pantalla, el micrófono, o ciertos sensores como el acelerómetro, el de temperatura, de presión, etc).

Podríamos sustituir completamente el emulador por un dispositivo físico, pero no siempre es lo más cómodo para el desarrollo. Además, reduciríamos su vida útil al darle un uso tan intensivo (el dispositivo ha de estar todo el rato enchufado al ordenador, por lo que la batería puede verse afectada en su autonomía; así como tener que estar continuamente instalando y desinstalando aplicaciones reducen los ciclos de uso de la memoria; además de que las aplicaciones en desarrollo al no estar optimizadas al ejecutarse podrían producir un calentamiento excesivo en el dispositivo con el consecuente efecto para los componentes).

¿Cómo configuro nuestro dispositivo para depurar aplicaciones?

1. Vamos a la “Ajustes”
2. Si tenemos **Android 4.2 o superior** (en otro caso nos saltamos este paso y el siguiente) no aparecerán **las opciones de desarrollo, están ocultas.** Para hacerlas aparecer **vamos a “Información del teléfono”**





3. (Este paso, al igual que el anterior, nos lo saltamos si tenemos una versión de Android inferior a 4.2) Ahí **pulsamos varias veces sobre “Número de compilación”** hasta que aparezca un mensaje que nos indique que se han activado las opciones de desarrollador. Con esto nos aparecerá en el menú anterior las opciones que necesitamos, por lo que volvemos atrás

4. Elegimos “**Opciones de desarrollo**” (en otras versiones de Android el nombre será parecido)





5. Es necesario **activar la opción de “Depuración USB”** (Si es una versión nueva de Android necesitaremos activar previamente las “**Opciones de desarrollo**”). Al activar la depuración USB tendremos que aceptar el mensaje. Un **aviso de seguridad: cuando no estemos desarrollando tenemos que desactivar las opciones de depuración USB**, y las opciones de desarrollo para los Android más modernos; de no hacerlo pondremos en riesgo nuestro dispositivo, pues se podrán instalar aplicaciones maliciosas sin nuestro consentimiento.

6. Ya podemos enchufar nuestro dispositivo al ordenador. Aunque existen varias maneras lo mejor es por cable USB.

¿Cómo configuro nuestro ordenador?

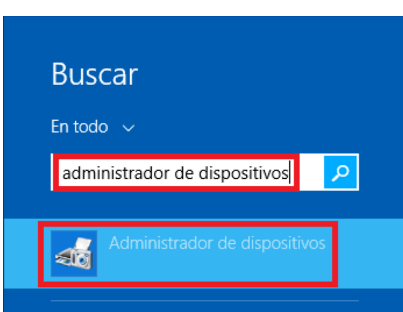
En nuestro ordenador lo que tenemos que hacerle es instalarle los drivers adecuados de nuestro dispositivo. Al enchufarlo por USB en modo desarrollo Windows nos instalará unos drivers genéricos o cono un poco de suerte los oficiales. Estos drivers lo más seguro es que nos sirvan, aunque no siempre es así o no suelen ser los más adecuados.

Los mejor es ir a la página del fabricante y descargarnos los drivers apropiados a nuestro dispositivo. A continuación pongo una lista de dónde encontrar algunos (Nota: como nos patrocina ninguno ponemos los que nos da la gana en estricto orden alfabético y los quitaremos de la misma manera; como este libro está diseñado para desarrolladores Android facilitaremos una gran lista de los drivers para los dispositivos más populares):

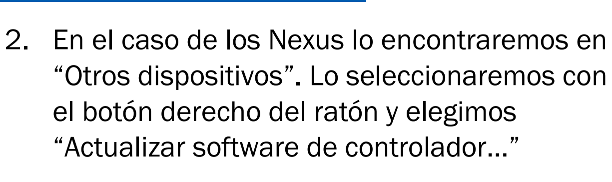
- **Acer**: http://www.acer.com/worldwide/support/index.html
- **Asus**: http://www.asus.com/es/support
- **Dell:** http://www.dell.com/support/drivers/us/en/19/ProductSelector
- **Google** (se explica la instalación en la siguiente pregunta)**:** http://developer.android.com/sdk/win-usb.html
- **HTC**: http://www.htc.com/www/support/
- **Huawey:** http://consumer.huawei.com/en/support/manuals/index.htm
- **LG**: http://www.lg.com/es/posventa/soporte-telefono-movil
- **Motorola:** https://motorola-global-portal.custhelp.com/app/home/action/auth
- **Samsung**: http://www.samsung.com/es/support/usefulsoftware/KIES/JSP
- **Sony:** http://developer.sonymobile.com/downloads/drivers/
- **Sony Ericsson**: http://www.sonymobile.com/es/tools/sony-ericsson-drivers/

¿Cómo instalo el driver genérico oficial de Android para diversos dispositivos?

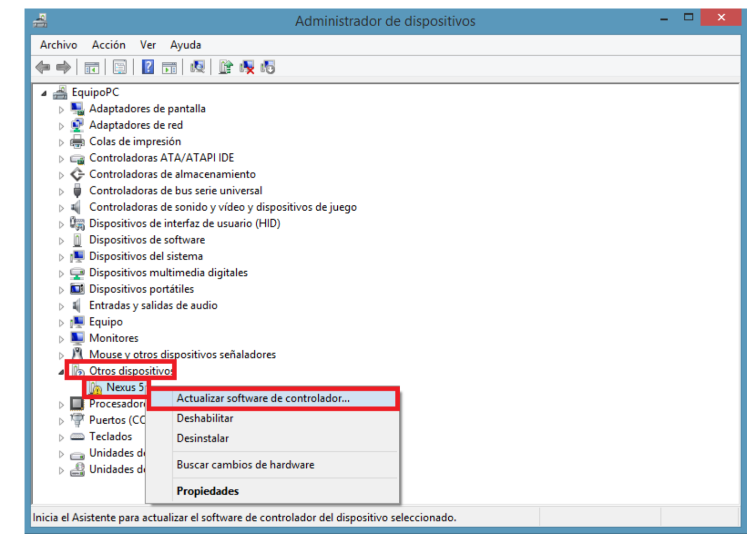
Este driver oficial de Google no solo sirve para que el ordenador reconozca únicamente a los dispositivos Nexus, sino también otros muchos cuyo sistema operativo no haya sido demasiado modificado y su Hardware se adapte a las especificaciones de Google (A otros dispositivos no les servirá). Lo que tenemos que hacer es lo siguiente:

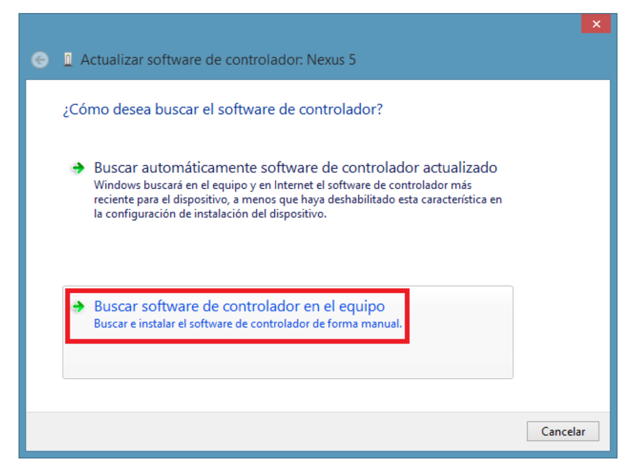


1. En Windows ir al "Administrador de dispositivos" (La imagen muestra un ejemplo del buscador de Windows 8)

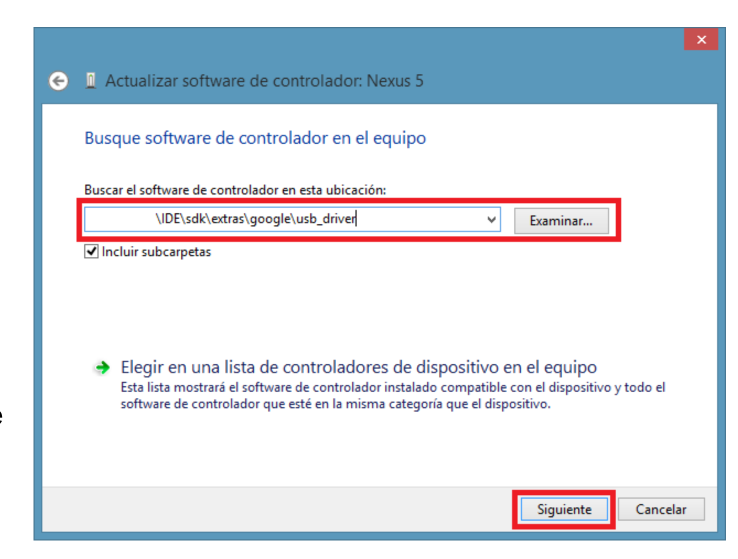


2. En el caso de los Nexus lo encontraremos en "Otros dispositivos". Lo seleccionaremos con el botón derecho del ratón y elegimos "Actualizar software de controlador…"





3. En la ventana que se nos abre seleccionaremos “Buscar Software de controlador en el equipo”



4. En la siguiente ventana lo buscamos en donde hayamos descargado el SDK de Android en la carpeta: sdk/extras/google/usb_driver



5. Se nos preguntará que si lo queremos instalar, le decimos que “Instalar”

6. Con esto ya lo tendremos instalado y nuestro dispositivo reconocido

¿Ya está todo listo para poder lanzar aplicaciones a mí dispositivo?

Si el dispositivo está es una versión anterior a Android 4.2 ya estará todo.

Por el contrario, si se dispone de Android 4.2 o superior necesitaremos pasar por otra medida de seguridad que será desde el **dispositivo permitir la depuración USB de un ordenador en concreto** (lo tendremos que hacer una vez para cada ordenador en el que queramos desarrollar en Android). Para ello podemos marcar la casilla de “Permitir siempre desde este ordenador” y pulsamos “Aceptar”. Ahora sí que estaremos listos.



Acelerar el emulador de Android

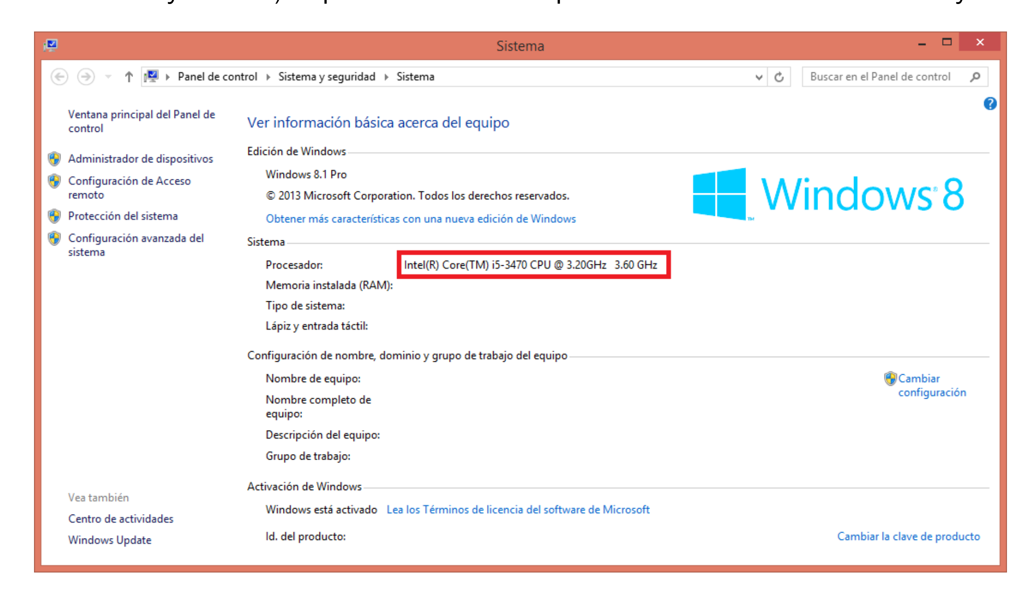
Es posible que notes que el emulador de Android no va tan rápido como nos gustaría. Una solución pasa por utilizar directamente un dispositivo móvil, pero en vez de esto existe otra solución. Usar una tecnología de Intel que acelera notablemente al emulador.

Evidentemente, no es necesario utilizar este emulador veloz para desarrollar en Android, pero sí es mucho más cómodo. Si tienes la posibilidad de usar este emulador te lo recomiendo, notarás un flujo más suave en el desarrollo de aplicaciones de Android.

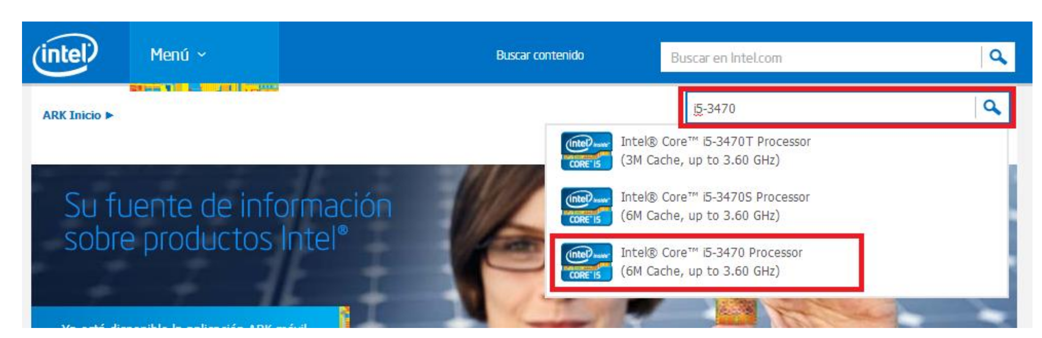
Lamentablemente no todos los ordenadores son compatibles con esta tecnología. Para que sean compatibles han de tener un **procesador Intel** que soporte **tecnología de virtualización de Intel** (Intel Virtualization Tech).

¿Cómo saber si mi procesador es compatible?

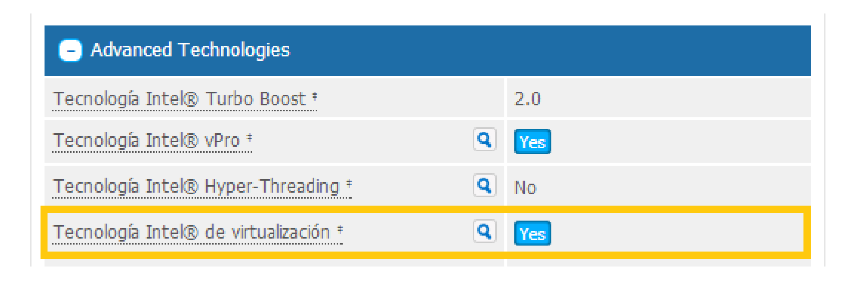
Prácticamente todos los procesadores Intel que se venden hoy en día disponen de esta tecnología. Para ver el modelo de nuestro procesador lo podemos ver en Windows en “Sistema” (puedes buscar en el buscador la palabra “Sistema” como se ya se vio). Aquí veremos en el apartado “Procesador” si es Intel y el modelo



Conociendo el modelo podemos saber si nuestro procesador es compatible. Puedes comprobarlo en la página http://ark.intel.com/es-es/. Introduce el modelo en el buscador de la web y elige tu procesador.



Si bajamos hasta el apartado “Advanced Technologies” podremos comprobar si disponemos de esta tecnología. Si vemos un “Yes” en “Tecnología Intel de virtualización” podremos utilizar el emulador veloz de Android.



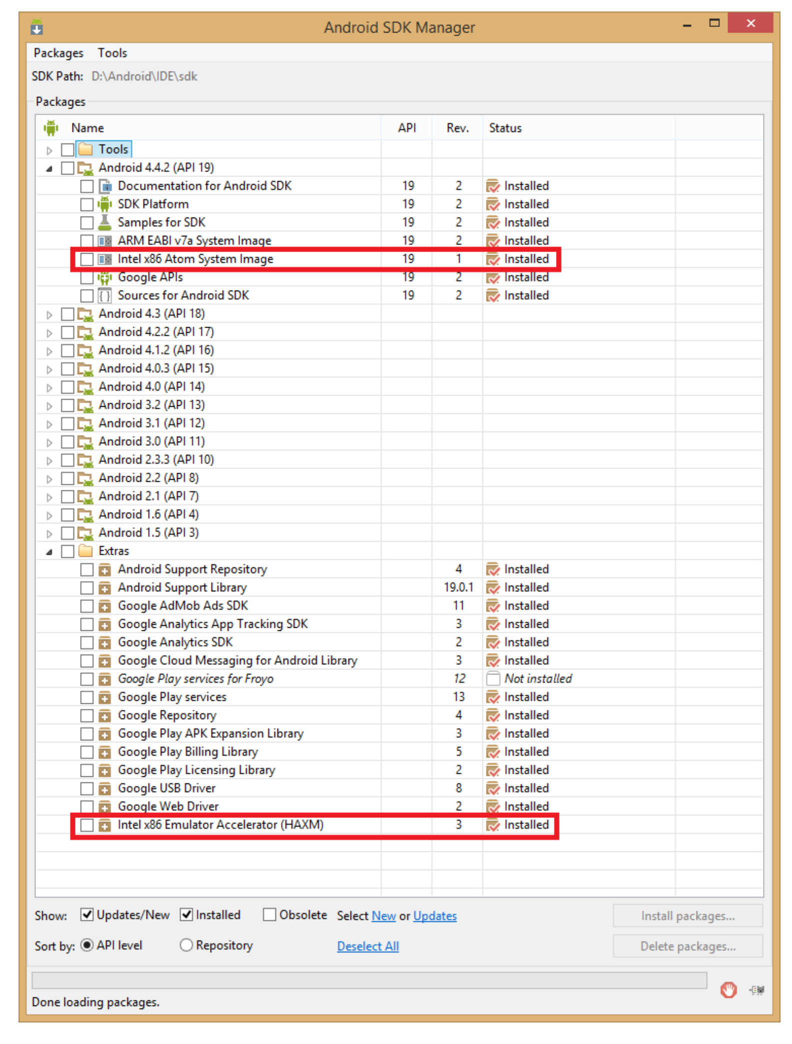
Advanced Technologies	
Tecnología Intel® Turbo Boost ‡	2.0
Tecnología Intel® vPro ‡	Yes
Tecnología Intel® Hyper-Threading ‡	No
Tecnología Intel® de virtualización ‡	Yes

¿Cómo configurar el emulador rápido de Android?

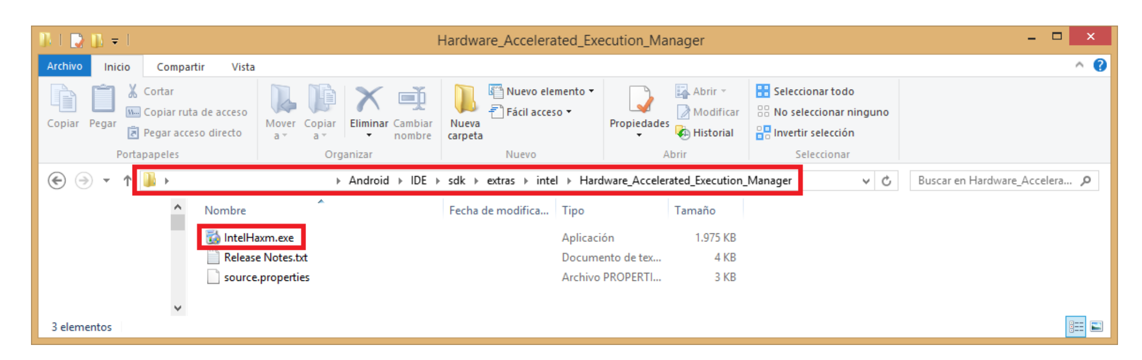
1. Vamos al SDK Manager de Android desde Eclipse.



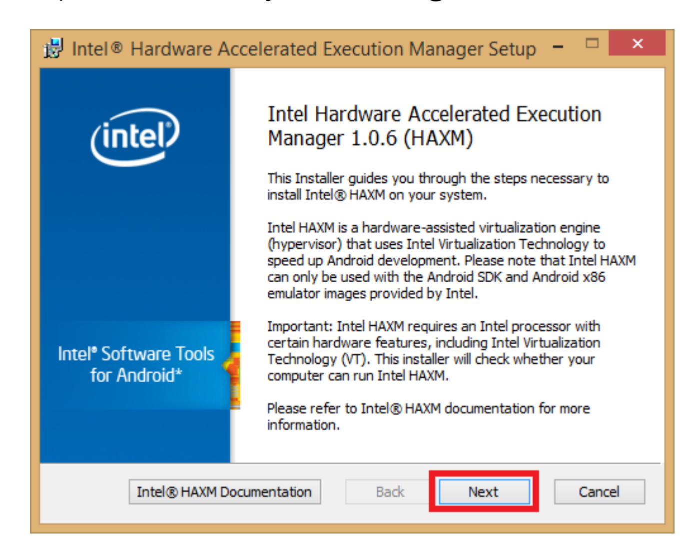
2. Para descargar los archivos necesarios (en caso de tenerlos descargados vamos al siguiente punto). Uno imprescindible es ir a la carpeta “Extras” y descargar “Intel x86 Emulator Accelerator (HAXM)”. Los otros que tenemos que descargar son las imágenes del sistema operativo preparadas por Intel, para ello vamos a las carpetas de Android de la versión que queramos descargar (en la captura siguiente vemos que estamos descargándola de “Android 4.2.2 (API 19)”) y descargamos las que se llaman “Intel x86 Atom System Image”.



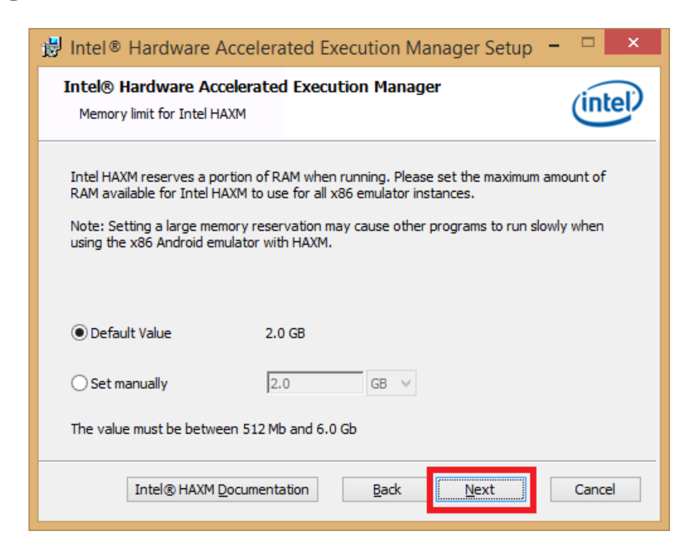
3. Después de la descarga vamos a la carpeta donde tengamos instalado el SDK de Android. Nos dirigimos a la ruta “IDE/sdk/extras/intel/Hardware_Accelerated_Execution_Manager”. Encontraremos el ejecutable “IntelHaxm.exe” que ejecutaremos.



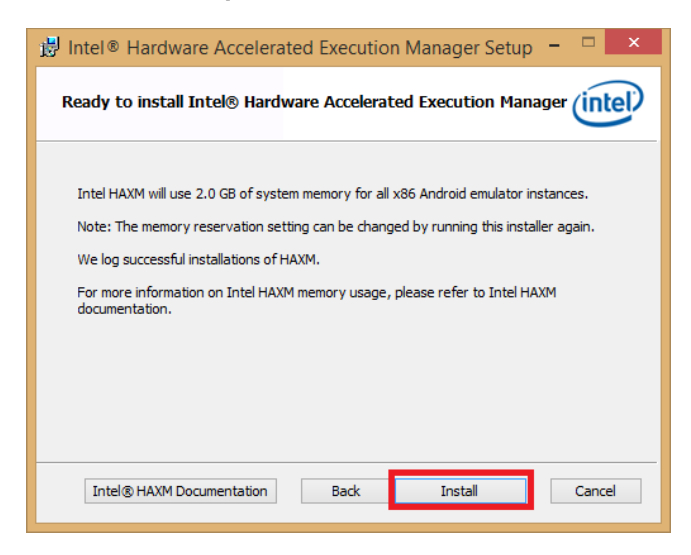
4. Naturalmente se nos abrirá un instalador. La primera pantalla nos dice esencialmente que solo va a funcionar si tenemos un procesador Intel y con tecnología de virtualización. Pulsamos en “Next”



5. La segunda ventana nos preguntará que cuánta memoria RAM de nuestro ordenador vamos a ceder al emulador. Aquí hay que tener en cuenta que cuando se lance el emulador la va a consumir (A más RAM que le permitamos teóricamente irá más deprisa el emulador y más lento nuestro ordenador en general), con lo que se recomienda el valor por defecto que es un cuarto de la memoria del ordenador (para evitar problemas recomendamos que se pongan un mínimo de 2GB, pues hemos probado con menos y no siempre arranca). Seguimos en “Next”.



6. Casi hemos terminado cuando en la siguiente ventana pulsemos “Install”.







No asustarse, si nos aparece esta advertencia. Simplemente pulsamos en "OK"

Simplemente nos indica que nuestro ordenador cumple con todos los requisitos pero que no tenemos activada la Tecnología de Virtualización. Para esto tendremos que ir a la Bios y activarla (lo explicamos un poco más adelante).

7. Terminamos la instalación pulsando "Finish".
8. Nos lo haya pedido o no tenemos que reiniciar nuestro ordenador en este momento.

¿Cómo activo la tecnología de virtualización del procesador?

Solo es necesario si nos apareció el anterior mensaje de supuesto error, o si queremos asegurarnos de que esté activada.

Nota: el sistema operativo de las Bios varía de una placa base a otra pero son parecidos, por lo que puede no parecerse a las imágenes aquí expuestas. Debido a la sensibilidad de esta parte del ordenador, recomendamos no tocar nada más de lo aquí explicado, o no hacerlo si no se tiene la total seguridad de que se está tocando ya que podemos estropear algo. Desde luego no nos hacemos responsables de esto, pese a que el proceso es muy sencillo y carente de riesgos si se hace bien.

1. Para ello a la que se inicia el ordenador, en la pantalla de la placa base pulsamos o la tecla "Suprimir", la de "Escape", o la que nos indique esta pantalla para entrar en la configuración de la Bios.
2. Aquí dependerá un poco del sistema de la Bios que tengamos. Pongo un par de ejemplo de dos Bios diferentes:
   a. En la siguiente captura vamos al apartado llamado "BIOS Features" y activamos (ponemos el valor a "Enable") el que se llama "Intel Virtualizacion Technology".

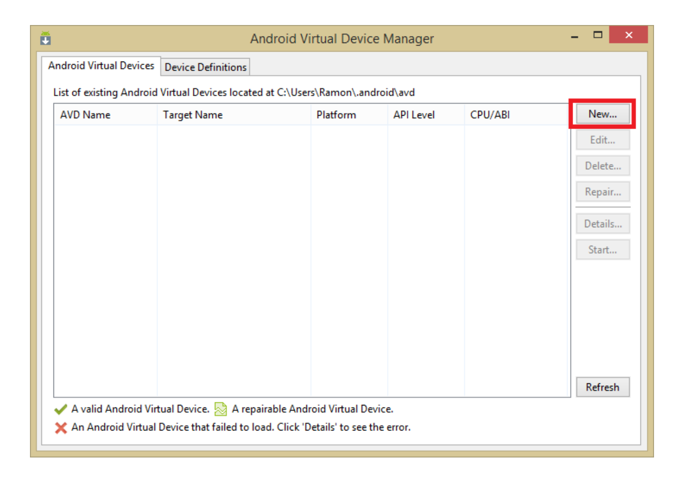
   b. Otra imagen de ejemplo de otra Bios diferente, en el que hay que ir "Advanced" luego a "CPU Configuration" y ahí activar el que pone "Intel(R) Virtualization Tech"
3. Para terminar guardamos y salimos, pulsando F10 y aceptando el mensaje

¿Cómo creo y utilizo un emulador con esta tecnología tan veloz?

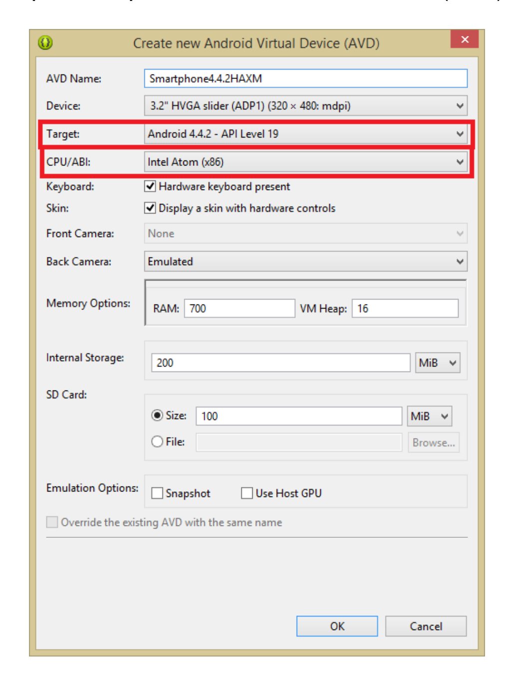
1. Cuando arranque el ordenador volvemos a Eclipse. Vamos al “Android Virtual Device Manager” para configurar un emulador.



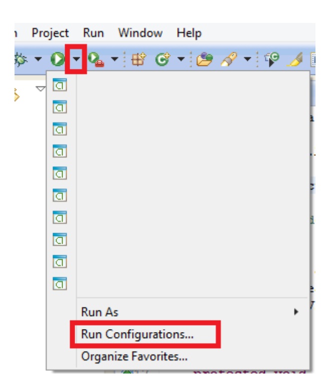
2. Aquí creamos un nuevo emulador pulsando “New…”



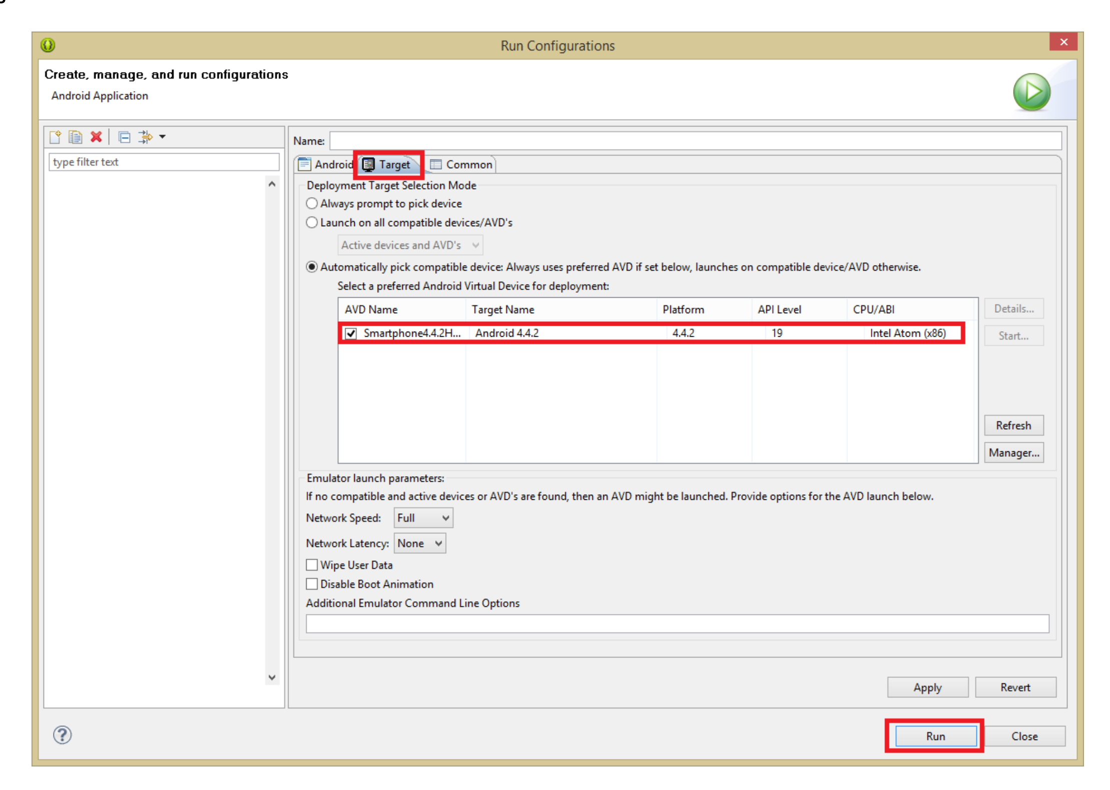
3. Entre todos los parámetros de configuración los verdaderamente interesantes ahora son el “Target” para seleccionar una versión del sistema operativo en la que nos hayamos descargado la imagen de Intel (No sirven los que empiezan Google APIs (Google Inc.), sino los otros). Y sobre todo el apartado de “CPU/ABI” donde seleccionaremos para que nos procese todo “Intel Atom (x86)”.



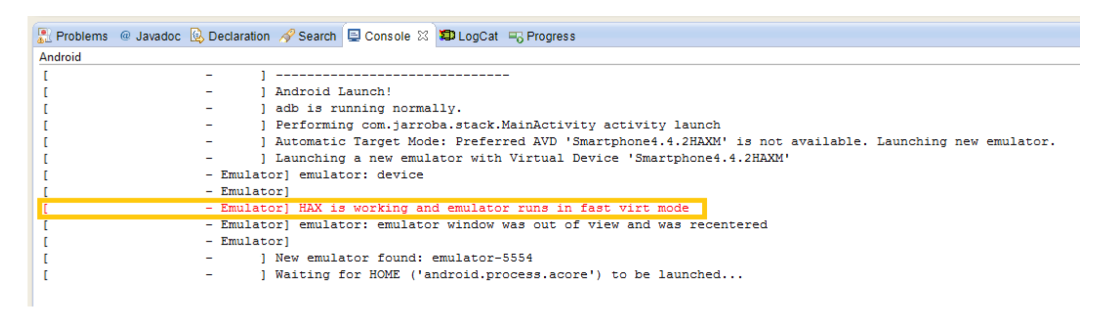
4. Ahora vamos a configurar lanzar el emulador que hemos configurado. Para ello vamos a Eclipse y pulsamos en la flecha negra que está justo a la derecha de a la flecha verde llamada “Run Stack”, ahí seleccionamos “Run Configurations…”



5. En la ventana que se nos abre vamos a “Target”, elegimos nuestro emulador que hemos configurado, y lo ejecutamos con “Run”.



6. El emulador se nos abrirá en unos segundos y se nos mostrará en la “Console” el siguiente mensaje diciendo que todo ha ido bien: “HAX is working and emulator runs in fast virt mode”



Me aparecido un error y tengo Windows 8.1

Puede que la consola no nos muestre el anterior mensaje sino este otro que es un mensaje de error, que significa que no está funcionando el emulador:

emulator: Failed to open the HAX device!
HAX is not working and emulator runs in emulation mode
emulator: Open HAX device failed

Para corregir esto tenemos que descargar el parche para Windows 8.1 de http://software.intel.com/en-us/articles/intel-hardware-accelerated-execution-manager e instalarlo. Prueba de nuevo y ya verás como el error se habrá solucionado.

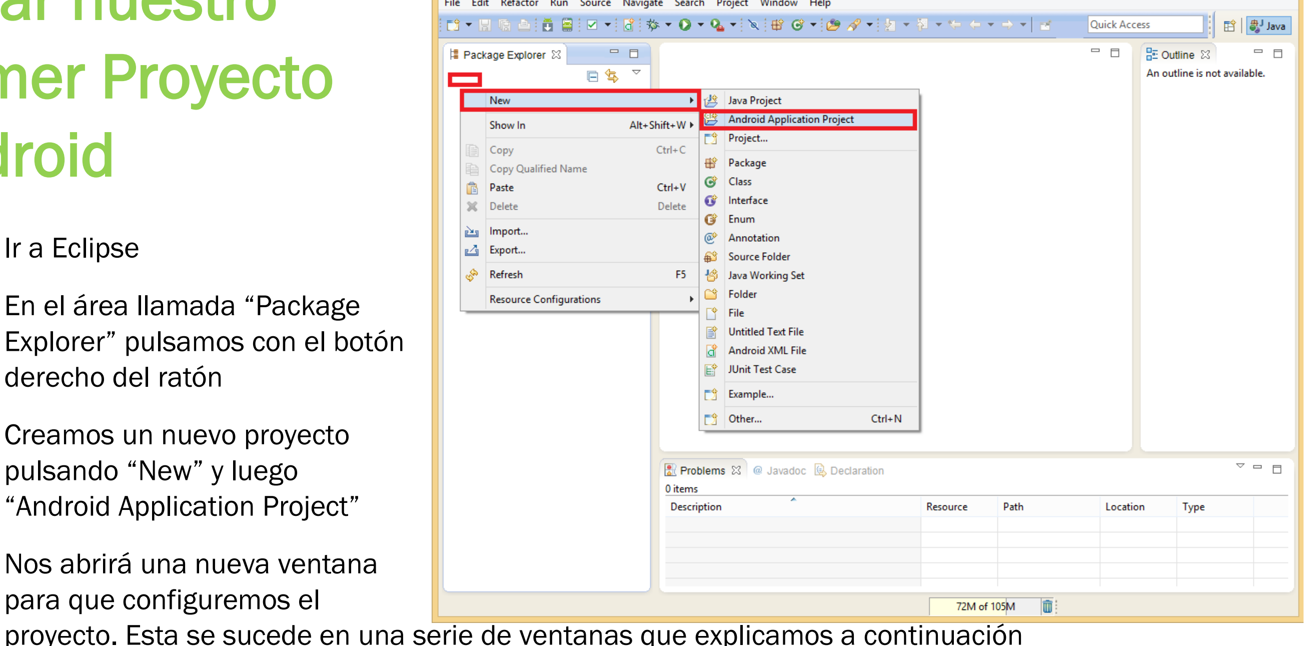
7. Ya podremos disfrutar de la velocidad punta de nuestro emulador Android

Referencias:

- http://jarroba.com/acelerar-el-emulador-de-android/
- http://developer.android.com/tools/devices/emulator.html#accel-vm
- http://software.intel.com/en-us/android/articles/speeding-up-the-android-emulator-on-intel-architecture

Crear nuestro primer Proyecto Android

1. Ir a Eclipse
2. En el área llamada “Package Explorer” pulsamos con el botón derecho del ratón
3. Creamos un nuevo proyecto pulsando “New” y luego “Android Application Project”
4. Nos abrirá una nueva ventana para que configuremos el proyecto. Esta se sucede en una serie de ventanas que explicamos a continuación



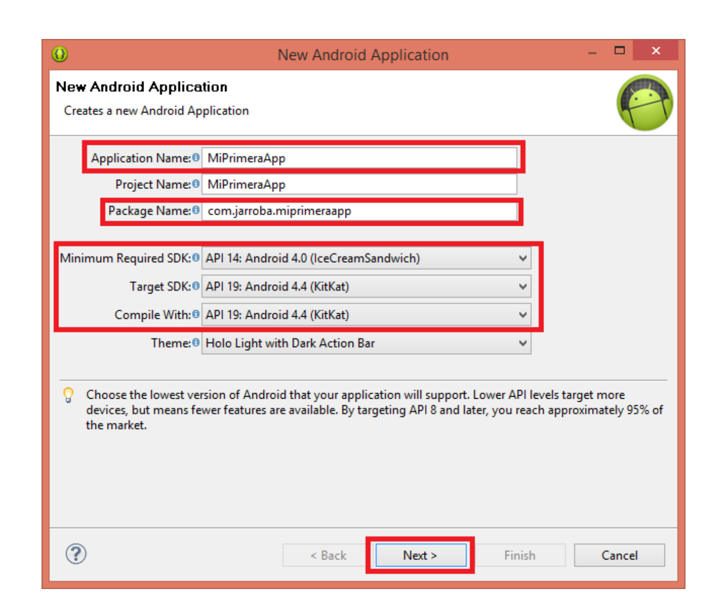
Configurar un nuevo proyecto

Vamos a explicar todo y a indicar cuales son las configuraciones indicadas para empezar a programar con Android. Decir que aunque haya muchas cosas, casi el 90% de las veces que creemos un proyecto constará únicamente de pulsar en los botones de “siguiente” de este asistente que vamos a explicar.

Paso 1: Crear una nueva aplicación Android

¿Qué contiene esta ventana?

- **Application Name**: Nombre que la aplicación tendrá en la Play Store
- **Project Name**: Nombre del proyecto para Eclipse
- **Package Name**: Paquete de nombre único que identificará a la aplicación durante toda su vida.
- **Minimum Required SDK**: La versión mínima de Android que la aplicación soportará
- **Target SDK**: Con que versión de las librerías de Android trabajaremos
- **Compile With**: Con qué versión se compilará. La versión más alta soportada



¿Qué debo tocar aquí para mi primera aplicación?

Únicamente has de **ponerle un nombre a la aplicación** (Application Name) y **modificar nombre del paquete** (Package Name) por defecto.

Indicar que estos dos no deberemos cambiarlos en un futuro, con lo que tenemos que estar seguros de qué nombre de aplicación y de paquete ponemos (realmente sí se puede, pero si los modificamos a mano y no lo hacemos bien vamos a tener problemas tanto con el proyecto en Eclipse como con la Play Store).

Para el nombre del paquete es recomendable poner una página web invertida (también podremos poner nuestro nombre, apellido u otro, siempre con letras minúsculas y sin espacios, tan solo separados por puntos). Por ejemplo de www.jarroba.com, el nombre del paquete será "com.jarroba" seguido del nombre de nuestra aplicación en minúsculas; como la aplicación se llama "MiPrimeraApp", el nombre del paquete resultará como "com.jarroba.miprimeraapp".

El **SDK mínimo** (Minimum Required SDK) **dependerá de la cuota de mercado que queramos acaparar o de las características avanzadas que queramos utilizar:**

- Llegar a los más usuarios posibles: "API 10: Android 2.3.3 (Gingerbread)" (menos no lo recomiendo, se pierde muchísimo tanto en avances como en seguridad, y no aporta muchos más usuarios)
- Máxima relación usuarios con características avanzadas: "API 14: Android 4.0 (IceCreamSandwich)"
- Lo último en tecnología Android sin importar la cantidad de usuarios: La más alta que haya

Para un desarrollador siempre es deseable una alta. Para empezar da un poco igual, pero en un futuro será recomendable usar de la 4.0 en adelante para poder usar características avanzadas.

Luego tanto el **SDK objetivo** (Target SDK) como el de **compilación** (Compile With) en mi opinión deberían de ser **los más altos que hayan** (En "Compile With" es recomendado seleccionar la más alta, salvo casos que se requiera que la app solo funcione en versiones antiguas del sistema operativo Android), para trabajar siempre con las bibliotecas de Android más avanzadas y que la compilación se realice de la manera más óptima.

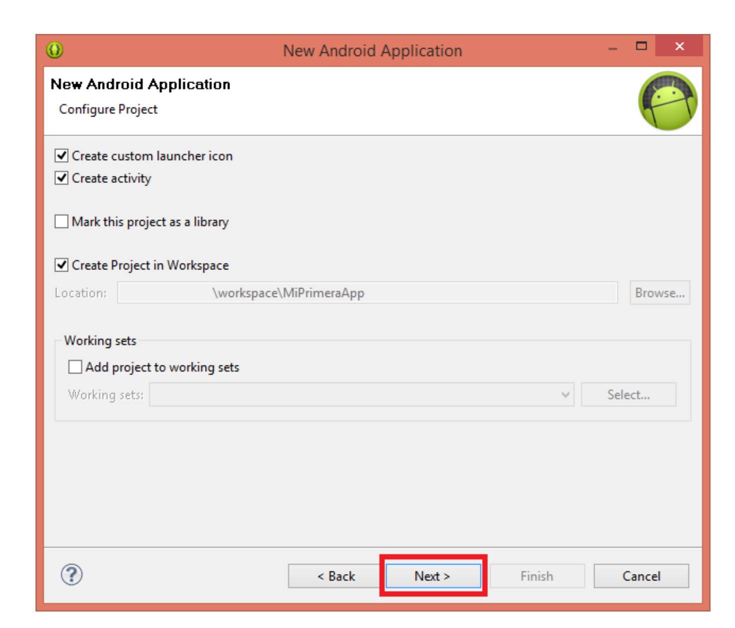
El tema (Theme) dejamos el que está.

Continuamos en "Next >"

Paso 2: Configurar el proyecto

¿Qué contiene esta ventana?

- **Create custom launcher icon**: crea un icono para la aplicación (Es decir, saltarse o no el siguiente paso)
- **Create activity:** crea la primera actividad con un hola mundo
- **Mark this Project as a library**: para que el proyecto sea una librería

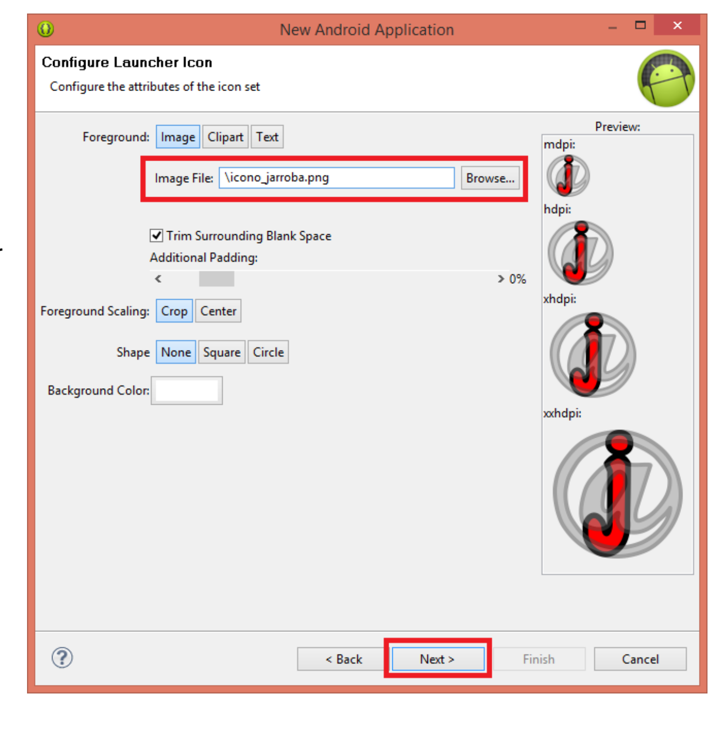


¿Qué debo tocar aquí para mi primera aplicación?

Nada. Tal como está pulsamos "Next >".

Paso 3: Configurar los atributos del icono

¿Qué contiene esta ventana?



- Esta ventana solo aparece si hemos seleccionado previamente **Create custom launcher icon**
- **Foreground**: El primer plano del icono puede ser una imagen que queramos, un dibujo prediseñado o un texto
- **Trim Surrounding Blank Space**: Recorta el espacio sobrante y lo rellena con la imagen
- **Additional Padding**: Relleno
- **Foreground Scaling**: Si queremos cortarlo o centrarlo
- **Shape**: La forma del icono que puede ser la que tenga el dibujo, en forma de cuadrado o de círculo
- **Background Color**: el color del fondo del icono

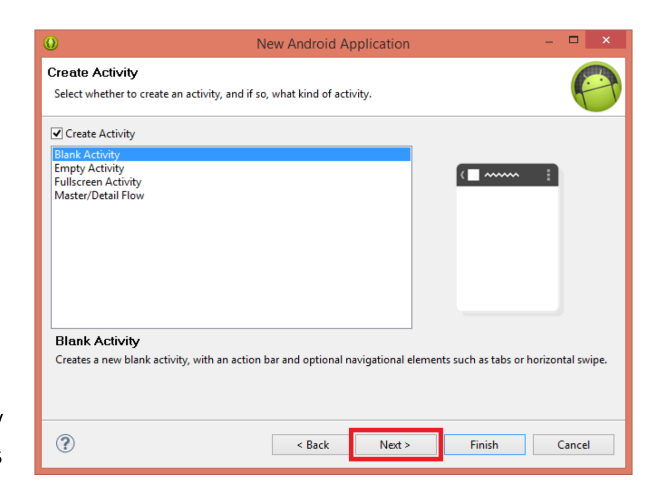
¿Qué debo tocar aquí para mi primera aplicación?

Añadir un **icono que queramos a la aplicación en donde pone "Image File"**. Aquí es interesante añadir un icono de alta resolución para que este asistente se encargue de escalarla y que no pierda demasiada calidad (Siempre será mejor diseñar un icono específico para cada resolución, en vez de usar este asistente, pero para facilitarnos la vida está bastante bien). Este icono (cuando creemos el proyecto lo veremos llamado como "ic_launcher.png" dentro de la carpeta "res/drawable") será el que aparezca tanto en el escritorio de Android (HomeScreen) para abrir la aplicación, como en la barra de título o de acciones (ActionBar); además de crearnos la imagen que necesitaremos para la Play Store (dentro del proyecto aparecerá con el nombre "ic_launcher-web.png")

Podremos configurar otras opciones, pero con esta es suficiente. Así que pulsamos en "Next >".

Paso 4: Seleccionar si crear una Activity y qué tipo

¿Qué contiene esta ventana?



Create Activity: si queremos que se cree alguna plantilla de aplicación. Estas son:

- **Blank Activity**: Crea una única actividad con Fragments y con Action Bar
- **Empty Activity**: Crea una Activity vacía
- **Fullscreen Activity**: Aplicación a pantalla completa, que al pulsar sobre la pantalla muestra la Action Bar y un botón en la parte inferior que permanecían ocultos
- **Master/Detail Flow**: Crea una lista con elementos seleccionables que se adaptan mediante Fragments tanto a Smartphone como a Tablets

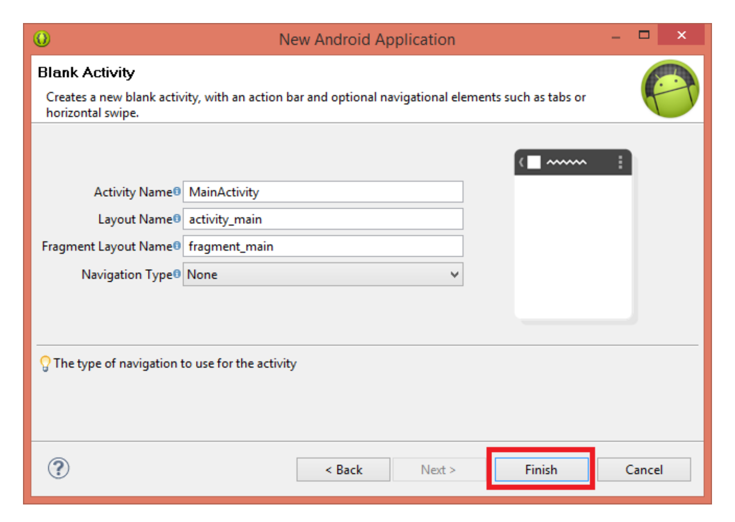
¿Qué debo tocar aquí para mi primera aplicación?

Nada. En un futuro será interesante elegir alguno de estos por ahorrarnos código, pero ahora nos interesa aprender cómo. Tal como está pulsamos "Next >".

Referencias:

- http://developer.android.com/tools/projects/templates.html

Paso 5: Seleccionar si crear una Activity y qué tipo



¿Qué contiene esta ventana?

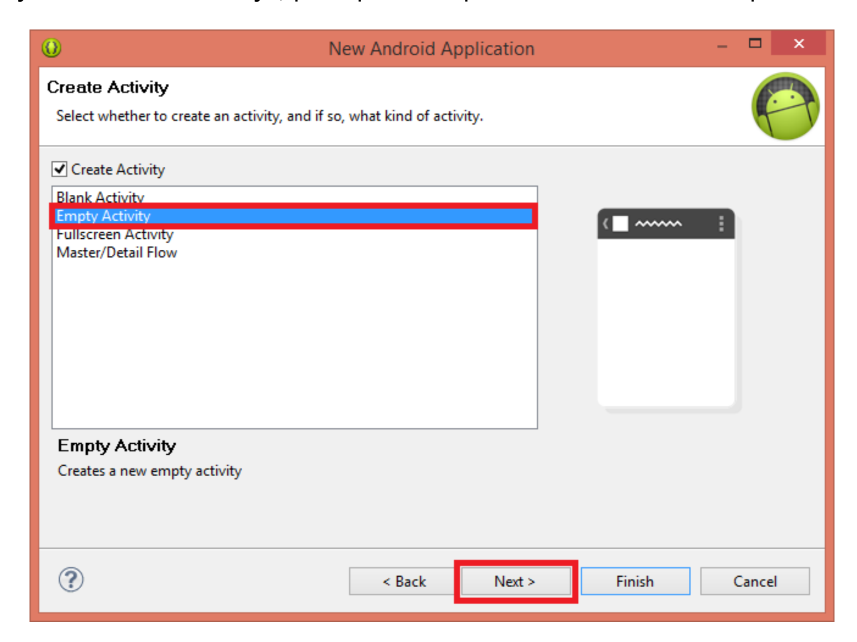
- **Activiy Name**: el nombre de nuestra Activity principal
- **Layout Name**: el nombre de la vista de diseño asociada a nuestra Acivity principal
- **Fragment Layout Name**: el nombre del primer Fragment que contendrá la Acitivity principal
- **Navigation Type**: Tipo de navegación entre las diferentes partes de nuestra aplicación para nuestra aplicación. Puede ser:
    - **None**: ningún tipo de navegación
    - **Action Bar Tabs (with ViewPager)**: Pestañas que se pueden alternar arrastrando el dedo por la pantalla
    - **Swipe Views (ViewPager)**: Páginas que se pueden alternar arrastrando el dedo por la pantalla
    - **Action Bar Spinner**: Navegación desplegable
    - **Navigation Drawer**: Navegación mediante menú lateral

¿Qué debo tocar aquí para mi primera aplicación?

Nada. Podremos cambiar el nombre de la Activity (Activity Name) o el nombre de su diseño asociado (Layout Name), pero para seguir este libro recomiendo no cambiarlo de momento, luego será muy fácil. Tal como está pulsamos "Finish". Habremos terminado de configurar nuestro primer proyecto de Android. Ahora solo queda programar.

Primer vistazo de un proyecto Android

Vamos a repetir el mismo proceso anterior pero en el paso “Paso 4: Seleccionar si crear una Activity y qué tipo” vamos a seleccionar “Empty Activity”. Y en el paso 5 pulsamos sobre “Finish”. (Cuando aprendamos utilizaremos casi para cualquier proyecto “Blank Activity”, pero para los primeros recomiendo aprender con “Empty Activity”)



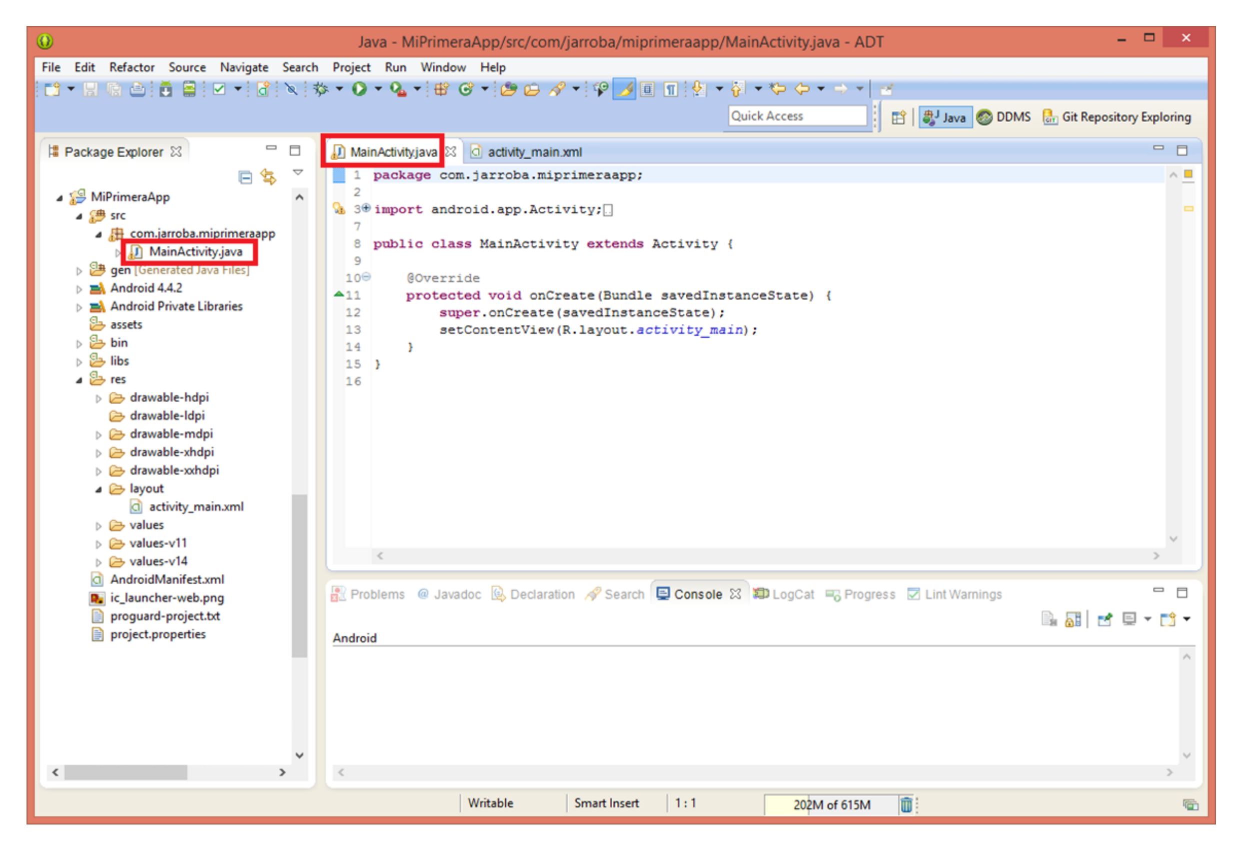
Al terminar el asistente anterior nos habrá quedado un entorno de trabajo como el siguiente.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${packageName}.${activityClass}" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

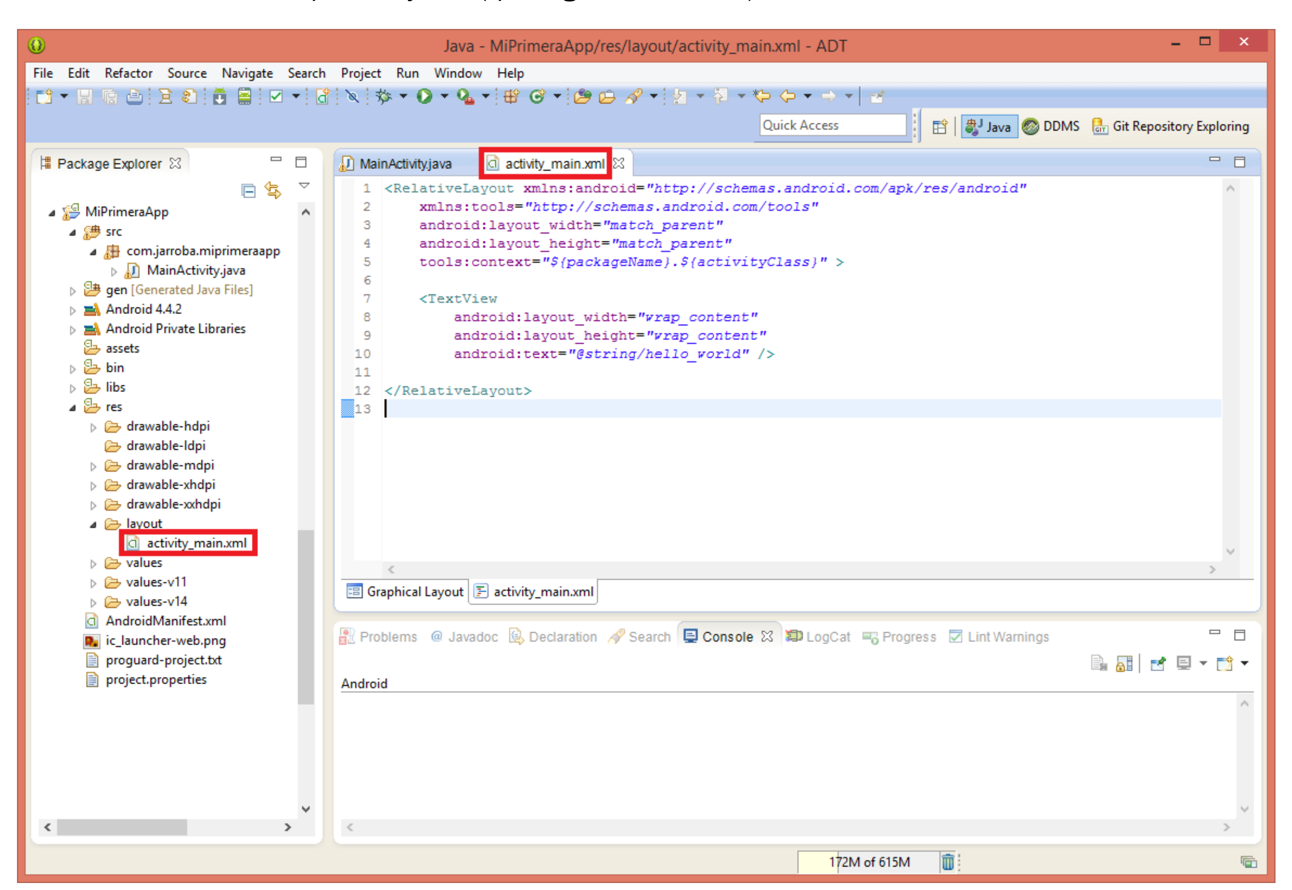
</RelativeLayout>
```

Vamos al fichero que se llama MainActivity.java. Para ello podemos hacerlo o bien pinchando en la pestaña abierta, o dentro del paquete Java en el mismo fichero.



Si te fijas está dentro de una carpeta llamada “src”. Aquí irá todo nuestro código Java.

¿Y el fichero activity_main.xml? Es para la vista de diseño, es decir, lo que ve el usuario: botones, textos, etc. Échale un vistazo y fíjate que se encuentra dentro de la carpeta “res” (de **res**ources que significa recursos), y más concretamente en la carpeta “layout” (que significa diseños).

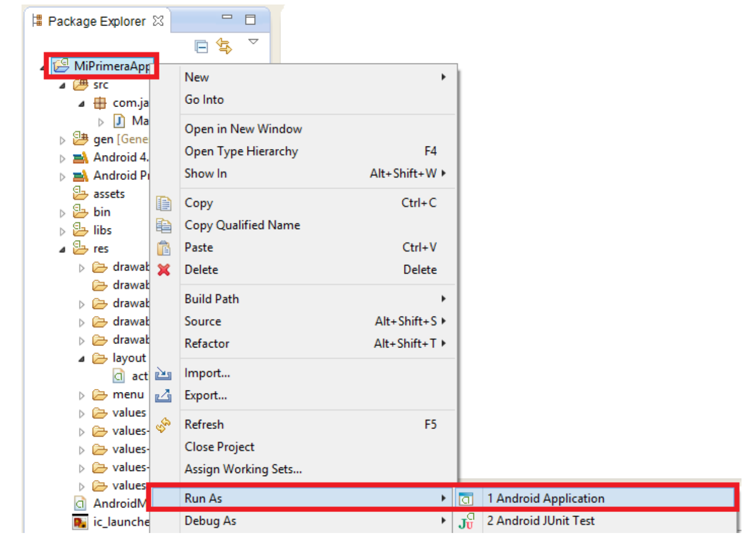


Probar nuestras aplicaciones

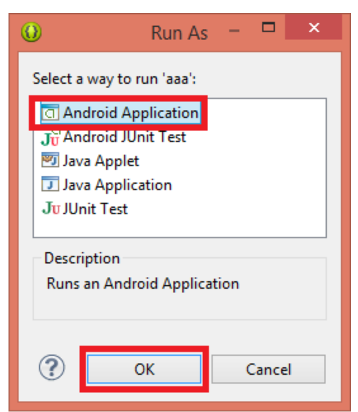
Para esto tendremos que tener un emulador configurado al menos con la misma versión o más alta del sistema operativo que hayamos elegido para compilar (en la primera ventana del asistente antes explicado en “Compile with”) o un dispositivo enchufado al ordenador en modo debug.

Tal como está la aplicación podemos ejecutarla:

- Si tenemos abierto un fichero Java de la aplicación y lo tenemos seleccionado podremos pulsar en **la flecha verde**



- Otro modo es pulsando con el botón derecho del ratón sobre la carpeta del proyecto de la aplicación y eligiendo “**Run As**” y “**Android Application**”



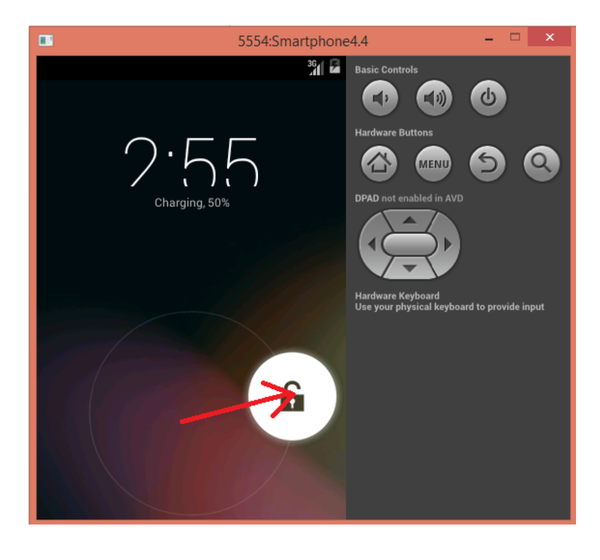
Al arrancar la primera vez el emulador para esta aplicación, aparecerá una pequeña ventana llamada “Run As”. Aquí seleccionamos “**Android Application**” y pulsamos “OK”.

Con esto la aplicación se lanzará en el dispositivo o en el emulador.

Si vamos a trabajar con el emulador y no lo tenemos abierto este se iniciará.

El emulador por primera vez

Cada vez que arranquemos el emulador tarda cerca de un minuto en arrancar la primera vez. Luego no hace falta que lo cerremos, podremos ejecutar tantas aplicaciones como queramos sobre este emulador abierto.



Siempre se inicia bloqueado (como cualquier dispositivo), para desbloquearlo basta con pinchar y arrastrar (variará dependiendo de la versión que hayamos arrancado de Android)

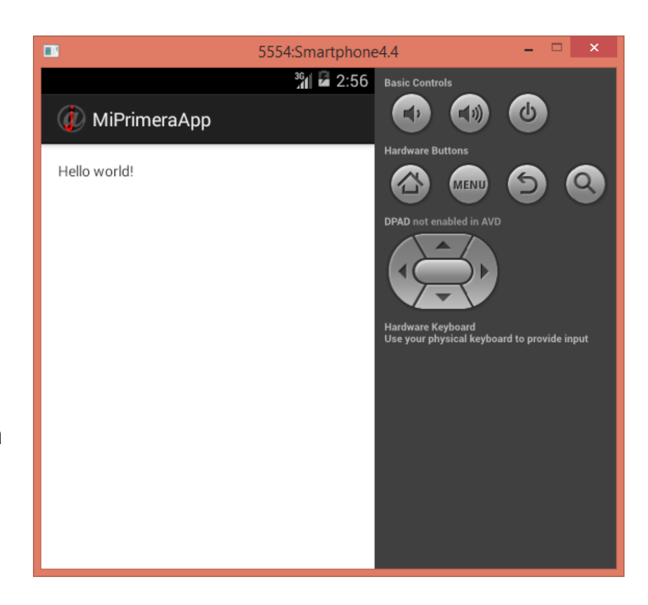
Los botones del emulador simulan los botones físicos de un dispositivo con Android. Estos son:

- Primera línea:
    - **Bajar volumen**
    - **Subir volumen**
    - **Apagar dispositivo**
- Segunda línea:
    - **Home** (volver al escritorio)
    - **Menú** (En desuso, podría desaparecer en versiones posteriores de Android)
    - **Atrás**
    - **Buscar** (No lo suelen tener los dispositivos de Android)

- Tercera línea con los **cursores físicos** (tampoco suele ser habitual en dispositivos Android)
- Luego hay una indicación que nos dice que para simular el teclado físico usemos nuestro **teclado del ordenador** (en versiones antiguas de Android aparecía un teclado en esta parte del emulador, pero es más cómodo el teclado del ordenador)

Si hemos ejecutado el “Hello Word!” anterior, lo veremos en pantalla del emulador como en la imagen de al lado.

Cada vez que lancemos la aplicación desde Eclipse, Android la instalará y lanzará automáticamente en este emulador.



Trazas y log

¿Son iguales las trazas de Android a las de Java?

Si hemos utilizado Eclipse para otros proyectos, seguramente estamos acostumbrados a utilizar el apartado “Console” para ver el volcado de logs, trazas y eventos. Programar en Android se programa con Java, pero a diferencia de Java como tal, que podemos crear trazas con:

Traza Java

```
System.out.print("Mi mensaje de la traza");
```

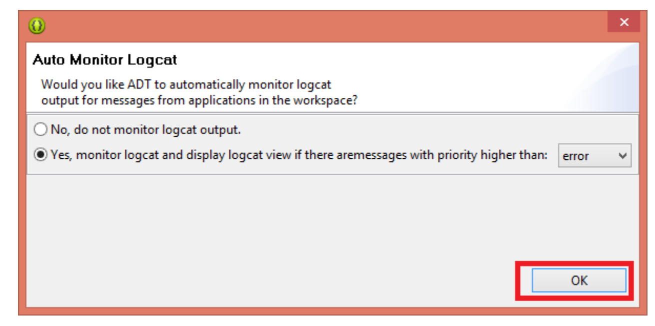
En Android no nos sirve, pues una vez en el emulador se le pasa el control a este y por tanto a Android y a su SDK, por lo que usaremos algo ligeramente diferente, como:

Traza Android

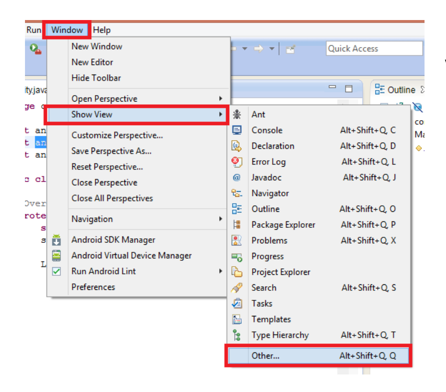
```
Log.v("Tag", "Mi mensaje de la traza");
```

¿Cómo abro el logcat?

La primera vez ejecutemos una aplicación -tanto en el emulador como en un dispositivo- nos aparecerá una ventana en Eclipse titulada “Auto Monitor Logcat”. Podemos pulsar en “Yes, monitor logcat and…”, aceptar la ventana y ya se abrirá solo el Logcat.



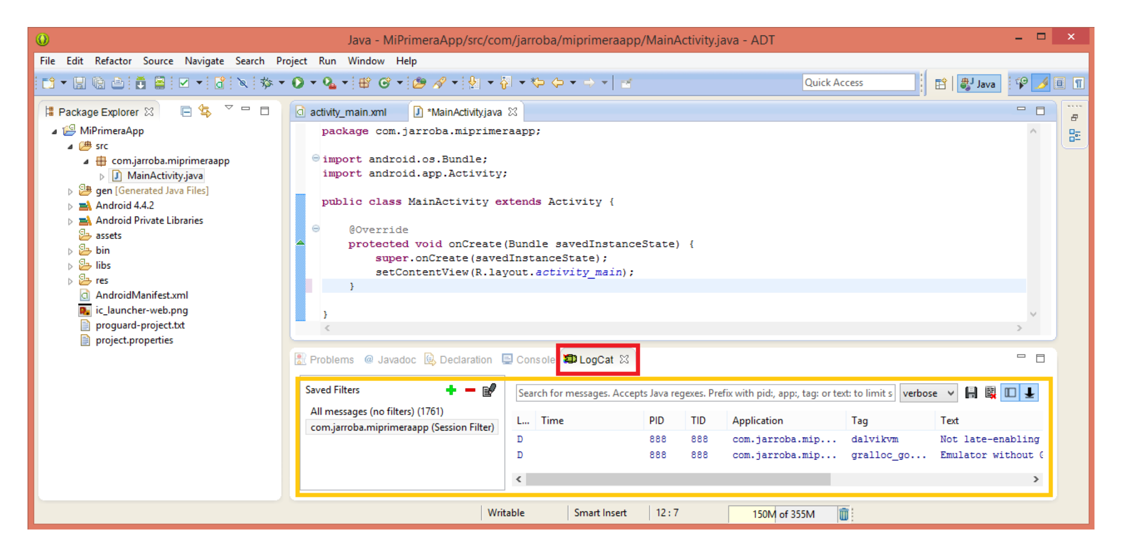
Si la hemos cerrado sin más o no nos ha aparecido, no pasa nada, es fácil de mostrar.



1. Vamos a la barra de herramientas de Eclipse a “Windows/Show View/Other…”

2. En la ventana que se nos abre, desplegamos la carpeta “Android”, elegimos “LogCat” y aceptamos

De cualquiera de las dos maneras se nos va a abrir el LogCat, que es donde se mostrarán las trazas.



¿Cómo se programan las trazas?

Esta traza de Android requiere que importemos la biblioteca: **android.util.Log**

Truco Eclipse: Importar todo

Cuando haya que importar una o varias bibliotecas en Eclipse –en vez de ir una a una- tan sólo tendremos que pulsar la combinación de teclado: **[Ctrl] + [Mayús] + [O]**

Como está puesto en el anterior código, todas las trazas comienzan con "Log" y luego un método que es una letra minúscula. Existen varias trazas en Android, estas se dividen principalmente por el nivel de la advertencia más básica hasta el error más grave. A continuación se muestran ordenados desde el más grave al menos:

- Error: Error que provoca un fallo de ejecución (el evento que ocasionó el error; como el típico NullPointerException por ser nula una variable donde no debe)

```
Log.e("Tag", "Mi mensaje );
```

- Warn: Advertencias (eventos que no van a provocar error pero que debieran revisarse; como que no se ha podido cargar una imagen y se ha devuelto una predeterminada)

```
Log.w("Tag", "Mi mensaje");
```

- Info: Información (normalmente datos de interés como estados, valores, etc; un ejemplo sería indicar el tamaño consumido de la memoria)

```
Log.i("Tag", "Mi mensaje");
```

- Debug: Depuración (para mostrar una serie de eventos de depuración y registrarlos; como el cambio del valor de una variable mientras se usa el programa, para estudiar su comportamiento)

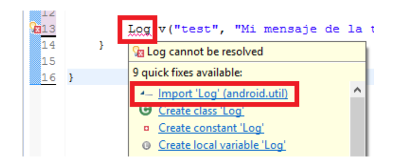
```
Log.d("Tag", "Mi mensaje");
```

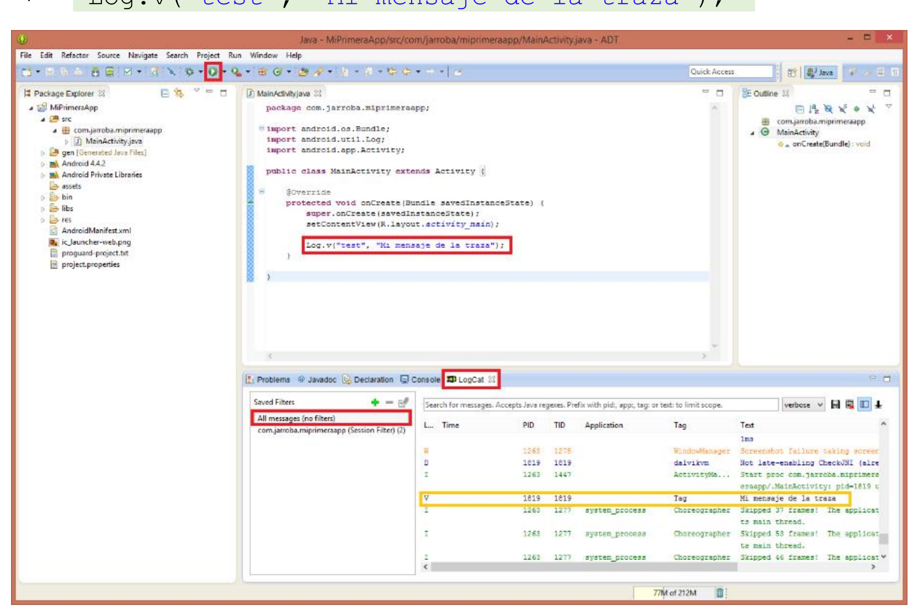
- **Verbose**: Detallar (cuando no sepamos que etiqueta usar o si las trazas no van a ser finales)

```
Log.v("Tag", "Mi mensaje");
```

¿Un ejemplo por pasos de cómo se programa una traza?

1. Añadimos dentro del método “onCreate” la traza que dice (fíjate en que el “Tag” ahora es “test” y al terminar lee la siguiente pregunta): `Log.v("test", "Mi mensaje de la traza");`
2. Aparecerá una línea roja debajo de la palabra Log
3. Situamos el cursor encima de la línea roja y seleccionamos del cuadrado que se nos abre: “Import ‘Log’ (Android.util)”



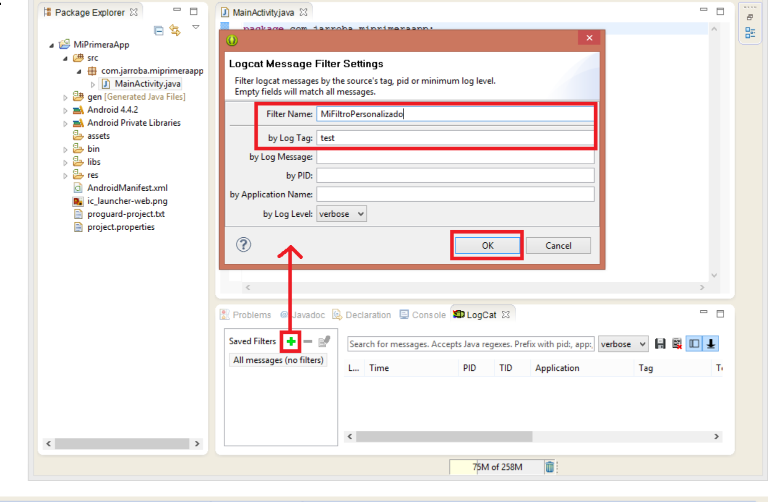


4. Ejecutamos la aplicación
5. Miramos en el LogCat nuestra traza

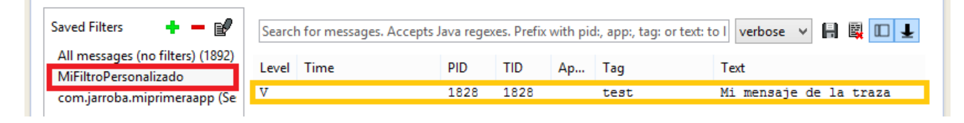
¿Cómo filtro los mensajes que salen en el LogCat para que no aparezcan tantos?

Para crear un filtro en el LogCat pulsamos el botón “+”. En el cuadro de dialogo que se abre, rellenar el campo “File Name” con el nombre que queramos que tenga el filtro. El resto de campos son optativos, aunque para empezar recomiendo rellenar el campo “by Log Tag”. Los campos que aparecen son los siguientes:

- **by Log Tag**: Etiqueta que pondremos para filtrar (recomiendo una palabra corta por comodidad). Por ejemplo, si escribimos por ejemplo “test”, pondremos: Log.v(“test”, “este mensaje se filtra por el TAG”);
- **by Log Message**: Filtrar por una parte del mensaje.
- **by PID**: Filtra por el identificador del proceso.
- **by Application Name**: Filtra por el nombre de la aplicación.
- **by Log Level**: Filtra por el nivel del Log (Dicho mal y pronto, filtra por colores).



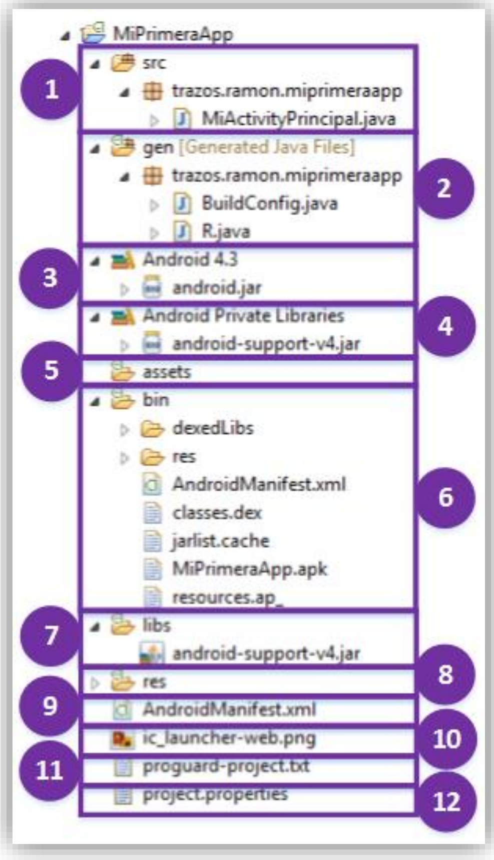
Una vez creado volvemos a ejecutar el programa. De este modo al volver a ejecutar se nos filtrará por el tag “test” que pusimos.



Referencias:

- http://jarroba.com/preparar-las-herramientas-necesarias-para-programar-en-android/
- http://developer.android.com/reference/android/util/Log.html
- http://developer.android.com/tools/debugging/debugging-log.html

Estructura de ficheros del proyecto



1. **src**: Contiene el código Java (código dinámico)
2. **gen**: Código generado automáticamente por el SDK. NOTA: No modificar manualmente.
    - **BuildConfig.java**: indica si la aplicación está en desarrollo
    - **R.java**: Clase automática que asocia el contenido de la carpeta res con identificadores para poder llamar a los recursos desde Java.
3. **Android x.y**: biblitecas oficiales y liberadas de Android, de la versión elegida
4. **Android Private Libraries**: Bibliotecas asociadas al proyecto que se ha añadido al path (es decir, para que se tenga acceso a las clases, con ello a los métodos y variables, para poder utilizarlos en el programa) desde la carpeta bin. Por defecto viene con una biblioteca que se llama "android-support-vX.jar", que incluye tanto soporte para versiones antiguas de Android como utilidades beta o que todavía no han sido liberadas
5. **assets**: Carpeta para introducir ficheros varios (Nota: las imágenes mejor guardarlas en la carpeta res), que permanecerán invariantes. Estos ficheros no serán asociados por R.
6. **bin**: se guarda el código compilado
7. **libs**: donde guardaremos las bibliotecas JAR en bruto (lo que es copiarlas de donde la tengamos, como puede ser el escritorio después de haberla descargado de Internet, y pegarla aquí dentro). Esta carpeta viene con una biblioteca de regalo "android-support-vX.jar" y que ya está añadida al path, como se indicó previamente.
8. **res**: carpeta de recursos de la aplicación (La explicamos con detalle en el capítulo de Recursos)
9. **AndroidManifest.xml**: Representa la información esencial de la aplicación que debe de conocer el sistema operativo, antes de que la ejecute. Entre otras cosas contiene:
    - Nombres de los paquetes de la aplicación.
    - Los componentes esenciales de una aplicación, que son: Activities, Services, BroadcastReceivers, y ContentProviders
    - Los permisos para acceder a las partes protegidas de la API o para interactuar con otras aplicaciones
    - Declara el nivel mínimo del API de Android que requiere la aplicación
    - La lista de librerías que están vinculadas
10. **ic_launcher-web.png**: Icono de alta resolución (512x512, 32-bit PNG con alfa, de tamaño máximo 1024KB) que aparecerá en Google Play
11. **proguard-Project.txt**: Configuración de ProGuard para que reduzca, optimice y ofusque el código.
12. **project.properties**: Contiene la configuración del proyecto y el destino para ser generado. Este fichero que se genera automáticamente, para modificarlo pulsar con el botón derecho sobre el proyecto y seleccionar "Properties".

Fundamentos de una aplicación

Teoría básica

Se explicarán una serie de puntos para aclarar algunos conceptos de Android, pero indagaremos en cada uno a medida que avances por las páginas del libro:

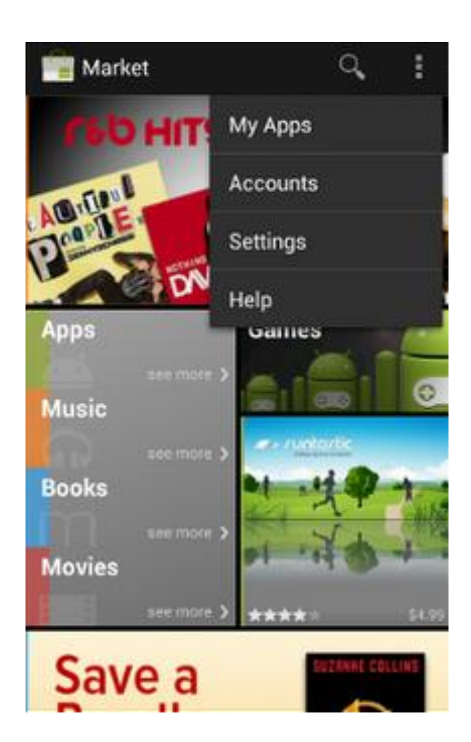
- **Instalación de una App**: Las aplicaciones Android se firman con un certificado y se empaquetan en un fichero .apk
- **Android tiene permisos multi-usuario**: Cada aplicación pertenece a un usuario (User ID Linux) diferente y solo puede acceder desde ese
- **Procesos**: Cada proceso se ejecuta sobre su propia máquina virtual. Por defecto, cada aplicación corre en su propio proceso Linux
- **Principio de privilegios mínimos**: una aplicación solo tiene acceso, por permisos asignados en el AndroidManifest.xml, únicamente a los componentes que requiere y nada más
- **Compartir datos entre Apps**: Se puede pedir permiso para acceder a los contactos del usuario, mensajes SMS, cámara, Bluetooth, etc
- **No existe el main()**: A una aplicación se puede acceder desde varios componentes si están dispuestos para ello
- **Componentes de una aplicación**: son bloques para construir aplicaciones Android con roles bien diferenciados. Todos los componentes son puntos de entrada a la aplicación (del usuario o del sistema). Cada componente tiene su propio ciclo de vida (son creados y destruidos)
- **Programación en Android**:
  - **Java**: para programar dinámicamente. Para la lógica del programa (por ejemplo, para hacer el qué es lo que ocurre cuando se pulsa un botón); siguiendo el patrón MVC (explicación del patrón MVC en http://es.wikipedia.org/wiki/Modelo_Vista_Controlador) para el controlador y el modelo.
  - **XML**: para programar estáticamente. Para la parte visual (por ejemplo, para dibujar el botón en pantalla); siguiendo el patrón MVC para la vista. También para información estática (como pueden ser los textos de la aplicación con sus idiomas).

Referencias:

- http://developer.android.com/guide/components/fundamentals.html
- http://developer.android.com/guide/topics/resources/available-resources.html

Componentes de una Aplicación

¿Cuáles son los componentes básicos?



Una aplicación puede estar formado por una o varios componentes básicos:

- **Activity**: Representa una pantalla con interfaz de usuario.
  - Ejemplo: de la App de correo, la pantalla de redactar un correo
- **Service**: Se ejecuta en segundo plano, para realizar operaciones de larga duración o para realizar trabajo proveniente de procesos diferentes.
  - Ejemplo: escuchar música sin tener la App en primer plano
- **Content Provider**: Permite a otras Apps consultar o modificar los datos almacenados en otra App.
  - Ejemplo: la App de contactos integrada en Android permite que otras App accedan a estos, como Whatsup que puede agregar contactos o consultarlos de ahí
- **Broadcast Receiver**: Responde a mensajes difundidos a todo el sistema. Se puede notificar al usuario mediante la barra de notificaciones del sistema.
  - Ejemplo: notificar a la App que la batería del dispositivo está baja; o recibir mensajes push y mostrar al usuario un mensaje de recibido en la barra de notificaciones

¿Es necesario declararlos en el AndroidManifest.xml?

Todos hay que declararlos en el AndroidManifest.xml

¿Se van explicar todos en este capítulo?

No, en este capítulo se explicará el primero: Activity. El resto se explicará a medida que se aprendan los principios necesarios para poder utilizarlos. Será entonces cuando demos explicación y ejemplos en profundidad

Activity

¿Qué es?

Algo enfocado a lo que el usuario puede hacer (de ahí que sea una “Actividad”). Por lo que interactúa con el usuario directamente. Es la ventana que llena a toda la aplicación, que contiene a la interfaz de usuario. Si cambiamos de ventana cambiaremos de Activity.

¿Se puede dividir en otras Activities más pequeñas contenidas en una Activity?

No, de eso se encargan los Fragments

¿Cómo se le asocia un Layout?

Justo después de la llamada al padre del método onCreate() con **setContentView(R.layout.mi_layout)**

¿De qué extiende la clase?

De Activity

¿Cómo se abre una nueva Activity?

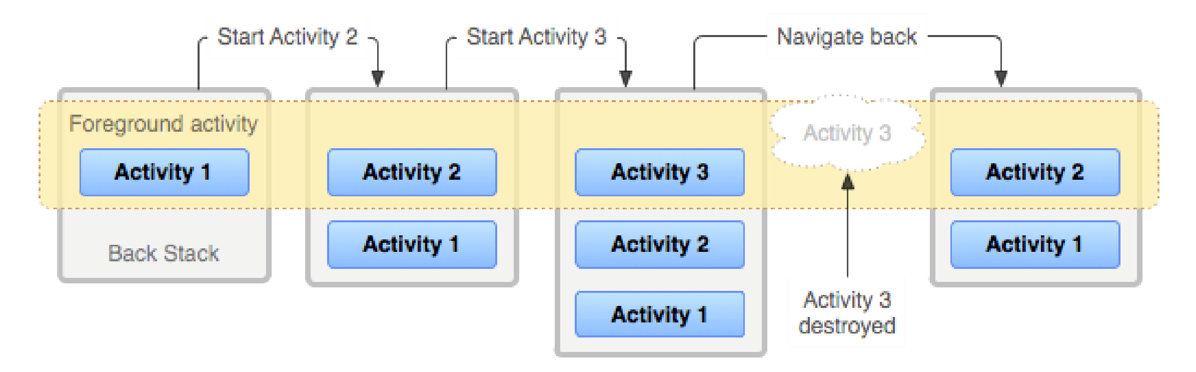
Con Context.startActivity(Intent miIntent) (Veremos esto en profundidad más adelante)

¿Cómo se declara en el AndroidManifest.xml?

Con el tag <activity>

¿Cómo se comporta una Activity cuando otra la sustituye?

Cuando una Activity se inicia, comienza en la cima de la pila de Activity. La Activity que está en la cima de la pila es la que se está ejecutando y mostrando al usuario. El resto de Activities quedarán por debajo en la pila. En el momento en que la Activity sobre la cima se desapile (deje de existir; por ejemplo al pulsar el botón de “atrás”), la que está por debajo será la que ahora ocupe la cima de la pila y por tanto la que estará al frente del usuario.

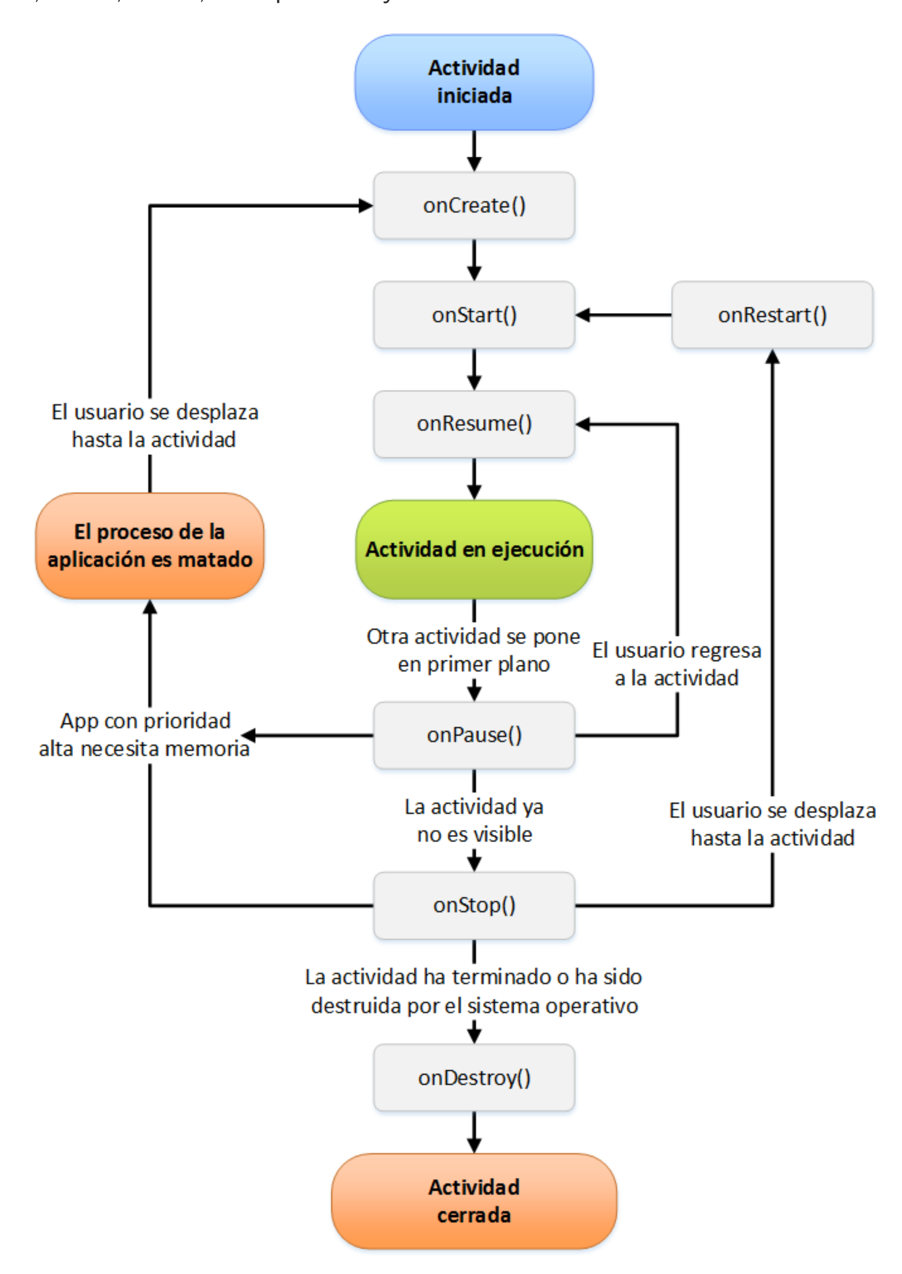


Referencias:

- http://jarroba.com/activity-entender-y-usar-una-actividad/
- http://developer.android.com/reference/android/app/Activity.html

Ciclo de vida de una Activity

Una Activity se caracteriza por tener un ciclo de vida. Se llama igual que el ciclo de vida de un ser vivo, ya que son semejantes: nace, crece, come, se reproduce y muere.



Supongamos que tenemos un en la mano un dispositivo con una aplicación iniciada. Por tanto habrá una Activity en pantalla, que podrá seguir varios caminos durante su vida, como:

- Arrancar la Activity: Pasará por Crear, Empezar y Continuar, para llegar a la ejecución normal.
- Usar de manera normal la Activity: estamos en la Activity propiamente, estamos en ejecución.
- Una ventana emergente se ha abierto: Pasará por Pausar.
- Cambiar a otra Activity o bloquear el móvil: Pasará por Pausar y Parar. (Nota aclaratoria: si se cambia a otra Activity pasa necesariamente por pausar y parar, no ha tenido que surgir una ventana emergente para pasar por pausar, si se cambia de Activity se pasa por ambos estados directamente; esto se da con otras acciones que pasen por varios estados).
- Apagar el móvil: Pasará por Pausar, Parar y Destruir.

El ciclo de vida completo que hemos visto en el gráfico anterior se traduce en el siguiente código (Todos los métodos son opcionales, aunque el onCreate() se recomienda usar siempre):

Activity con todos los métodos de su ciclo de vida

```
public class miActividad extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onStart() {
                super.onStart();
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onRestart() {
                super.onRestart();
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onResume() {
                super.onResume();
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onPause() {
                super.onPause();
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onStop() {
                super.onStop();
                //Nuestro código a ejecutar en este momento
        }

        @Override
        public void onDestroy() {
                super.onDestroy();
                //Nuestro código a ejecutar en este momento
        }
}
```

Nota y repaso del primer proyecto

Seguro que empiezas a unir piezas. Ya que existen cosas en común a la primera aplicación que hicimos. Por lo menos, la clase Java extendía de Activity y que tenía al método onCreate().

A continuación se muestra en detalle cómo funciona cada método:

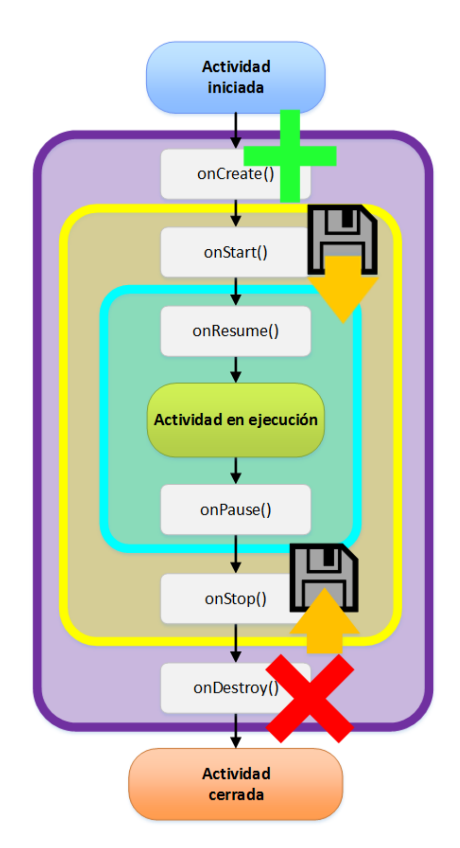
Método	¿Cuándo se llama?	¿Qué debería hacer?	¿Qué hay que tener en cuenta?
onCreate()	Al crearse	Crear views, unir datos a listas, etc	Entrega datos en un Bundle si la activity ha sido re-creada
onRestart()	Fue pausada y vuelve a ejecutarse		
onStart()	Al hacerse visible para el usuario	Recuperar el estado	
onResume()	Al comenzar la iteración con el usuario		En este momento la activity se sitúa en la cima de la pila
onPause()	Al perder el foco (cuando ya no interactúe con el usuario)	Se suele usar para guardar los cambios no guardados, para detener animaciones u otras consuman procesador	Cuando el onPause() termine, se realizará el onResume() de la nueva activity. Se recomienda código rápido
onStop()	Al no ser visible para el usuario	Guardar el estado	Ejemplo: otra nueva activity ha hecho su onResume() y a cubierto a esta
onDestroy()	Justo antes de ser destruida	Liberar sus recursos y limpiar su estado	Por la llamada a finish() o porque Android la haya matado

Estados de una Activity

Hay que tener en cuenta la probabilidad de que Android destruya nuestra Activity, en cara a salvar los datos y volverlos a cargar cuando la Activity vuelva a pantalla; lamentablemente esto hay que hacerlo manualmente.

Estado	Causa	Mantiene estado interno	Unido al gestor de ventanas	Riesgo de que Android la mate
Activa o en ejecución	Si está en primer plano (encima de la pila)	Sí	Sí	Bajo
Pausada	Si pierde el foco pero está todavía visible	Sí	Sí	Medio
Parada	Está completamente oculta	Sí	No	Alto
Matada	No existe	No	No	-

Vidas de una Activity



Sabiendo que puede morir nuestra Activity en cualquier momento. Es importante conocer las tres posibles clasificaciones de la vida, para poder crear y destruir, o guardar y cargar cuando sea necesario. Vidas:

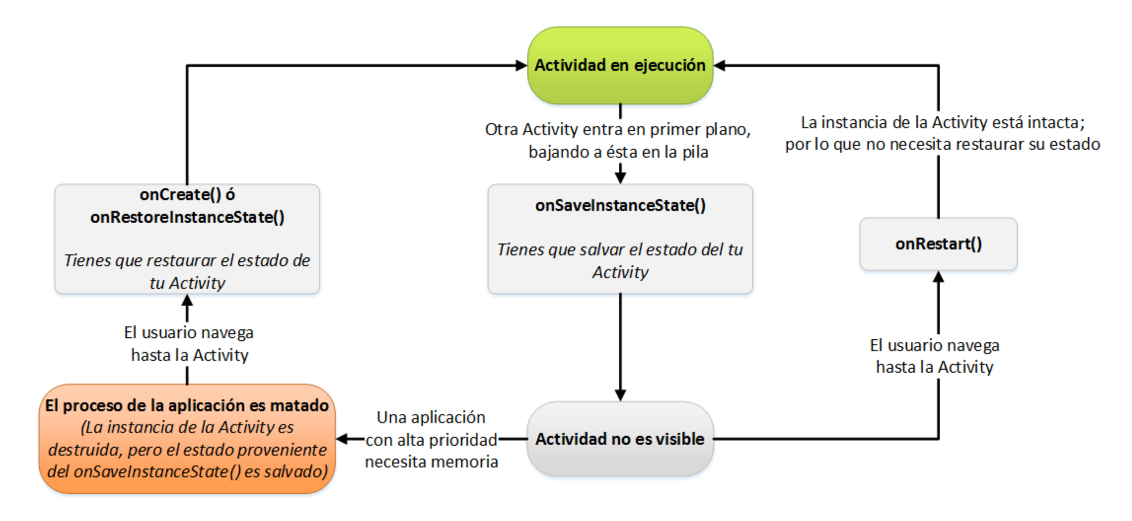
- **Completa**: desde el **onCreate()** donde crea el estado global, hasta el **onDestroy()** donde libera los recursos
- **Visible**: desde el **onStart()** donde se recupera el estado, hasta el **onStop()** donde se guarda el estado
- **Primer plano**: desde el **onResume(),** hasta el **onPause()**

Muertes de una Activity

Puede ser destruida

Si está:

- **En ejecución**:
  - Durante la ejecución de los métodos onPause(), onStop() u onDestroy().
  - Al cambiar algo de la configuración del dispositivo. Por ejemplo, al rotar la pantalla.
- **Si ha perdido el foco o no es visible**: Si Android requiere los recursos que está consumiendo



Tratar el estado ante la muerte y re-creación

Tendremos que:

- **Guardar el estado al hacerse no visible** (solo si tiene sentido ser re-creada): onSaveInstanceState(Bundle)
- **Recuperar el estado al re-crearse** (tiene que haber muerto previamente):
  - Al ser creada otra vez con onCreate (Bundle savedInstanceState): El Bundle será diferente de null
  - Después de onStart(): onRestoreInstanceState (Bundle savedInstanceState)

Ejemplo de Activity

Este cuadro con una flecha explicará cada parte de la estructura del ejemplo que viene a continuación. El resto de ejemplos en este libro siguen el mismo patrón

Lo que haremos

En esta parte titulada “Lo que haremos” explicaremos lo que vamos a hacer en el ejemplo y los pasos a seguir para llevarlo a cabo

Crearemos un nuevo proyecto vacío con “**Empty Activity**”, como ya vimos en

“Paso 4: Seleccionar si crear una Activity y qué tipo”.

En la clase Java “MainActivity.java” sustituiremos el código que hay.

Arrancaremos la aplicación como se describió en el capítulo previo.

Veremos en el LogCat las trazas que se han puesto al arrancar la Activity, y cuando salgamos de ella (pulsando la tecla de atrás).

Nota: En este ejemplo no tocaremos nada de la interfaz gráfica, solo es para ver algunos ejemplos del comportamiento de Activity.

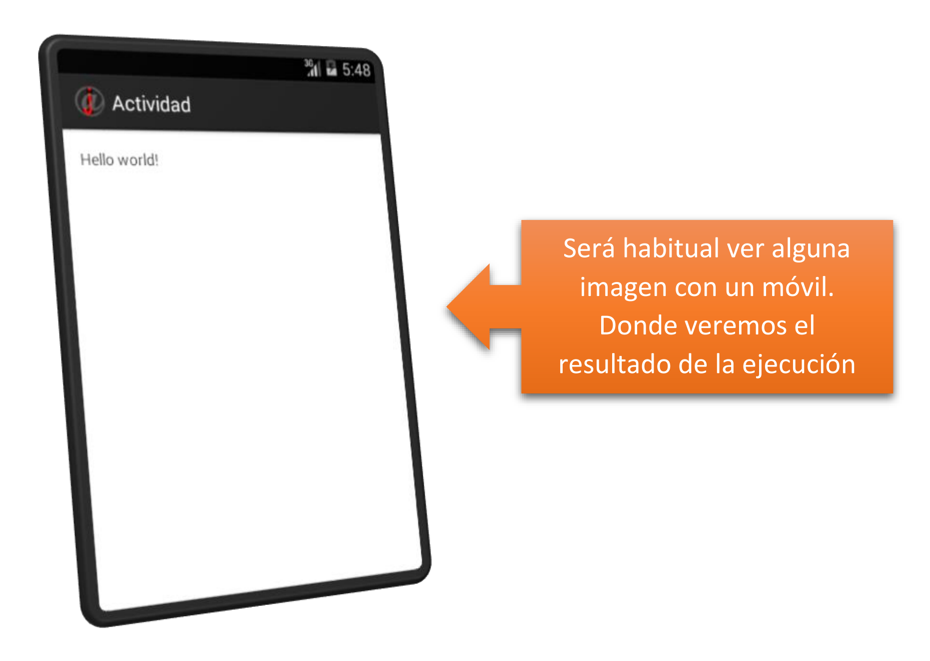


Ilustración 3 - Lo que se verá en pantalla, aunque no nos interesa en este ejemplo (solo puesto como demostración del tutorial)

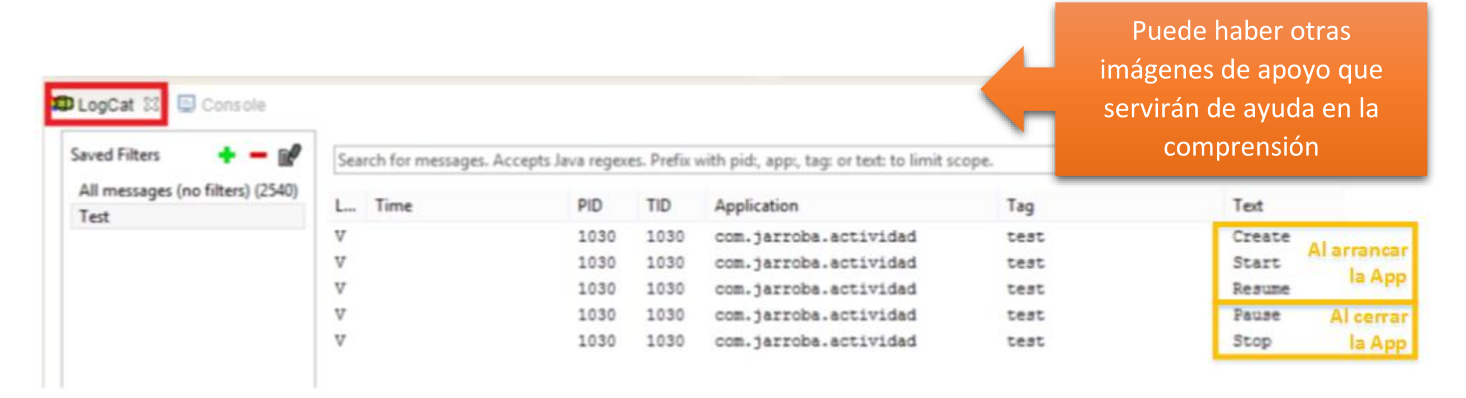
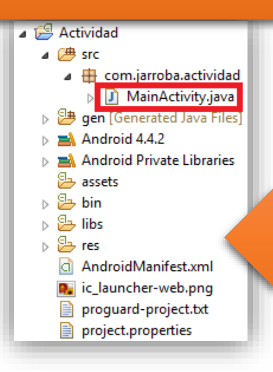


Ilustración 4 - Al ejecutar la aplicación o al terminarla nos mostrará en el Log el recorrido del ciclo de vida de Acvitity. Hay otros muchos, que iremos viendo

Proyecto

A partir de aquí te sugiero que abras Eclipse, e intentes hacer el ejercicio con la teoría previa (o que programes la solución mientras la entiendes). Lo que hagamos en un ejercicio habrá sido explicada con anterioridad, por lo que puedes hacerlo sólo fácilmente y sin mirar la solución



Al inicio del proyecto se muestra la estructura de ficheros que seguiremos con el código en Eclipse. Se marcara en rojo los ficheros modificados

MainActivity.java

```java
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

	private static final String TAG = "test";

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.fragment_main);
		Log.v(TAG, "Create");
	}

	@Override
	public void onStart() {
		super.onStart();
		Log.v(TAG, "Start");
	}

	@Override
	public void onRestart() {
		super.onRestart();
		Log.v(TAG, "Restart");
	}

	@Override
	public void onResume() {
		super.onResume();
		Log.v(TAG, "Resume");
	}

	@Override
	public void onPause() {
		super.onPause();
		Log.v(TAG, "Pause");
	}

	@Override
	public void onStop() {
		super.onStop();
		Log.v(TAG, "Stop");
	}

	@Override
	public void onDestroy() {
		super.onDestroy();
		Log.v(TAG, "Destroy");
	}

}
```

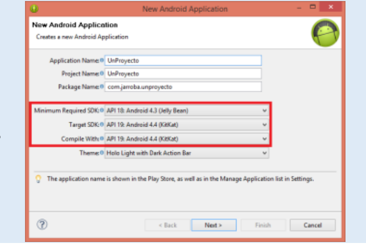
El nombre del fichero seguido de su código en rectángulos individuales. Utilizaremos el fondo verde para el código Java y el fondo amarillo para el código XML

Nota: No se incluirán las líneas "import" en el resto de códigos Java para aprovechar el espacio. En éste se incluyen por ser el primero. Si tuvieras dudas de cuál importar, puedes consultarlo en proyecto con el código que se facilita junto a este libro

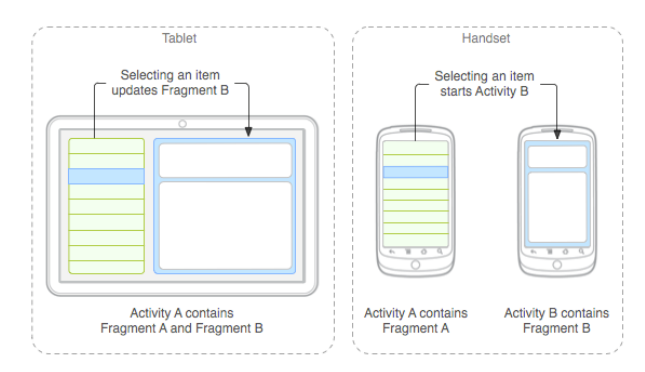
Algunas veces será interesante hacer "notar" algo. Lo mostraremos en un cuadro azul claro donde corresponda

¿En qué versión de Android trabajaremos en los ejemplos?

En este libro se explicarán las últimas versiones de Android. Por lo que no habrá código "Deprecated". Esto tiene sus ventajas (mayor estabilidad, menos consumo, lo último) y sus desventajas (en las versiones de Android antiguas o no funcionan o no van muy bien).
Para aprender recomiendo que tanto en "Minimum Required SDK" (se podrá bajar en un futuro), como en "Target SDK" y en "Compile With" establezcamos la última versión disponible.



Fragments



¿Qué es?

Es un comportamiento (no visible) o una porción de la interfaz gráfica (visible). Es decir, es un módulo que puede unirse a otros

¿Para qué surgieron?

Para que una misma aplicación pudiera ser multi-pantalla. Por este motivo, su mayor virtud es la reutilización de código.

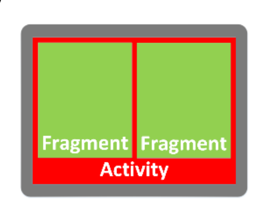
Supongamos que tenemos ciertas funcionalidades en varios Fragments. Para las diferentes Activitys, dependiendo del dispositivo, podremos acomodarlos de diferente manera. Hay que atender a que si nos pasamos poniendo Fragments para cierto tamaño de pantalla, se verá el contenido de cada Fragment demasiado pequeño y podría ser difícil de manejar; por el contrario, quedarnos supondría desaprovechar la pantalla. Unos ejemplos podrían ser (no es obligatorio cumplir con los siguientes ejemplos):

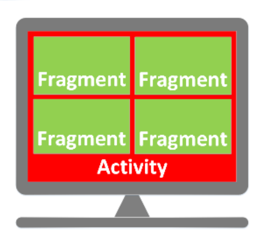


- Smartphone: en el que cabe 1 Fragment por Activity



- Tablet: en el que caben 2 Fragments por Activity





- SmartTV: en el que caben 4 Fragments por Activity

¿Qué ofrecen?

- Modularidad: Se pueden poner donde queramos
- Reusabilidad: Se pueden reutilizar tantas veces como se necesiten

¿Dónde se colocan los Fragments?

Siempre sobre una Activity. Dicha Activity se denomina “Activity contenedora” de Fragments

¿Tienen ciclo de vida?

Sí. El Fragment se mantendrá en ejecución mientras lo esté su Activity contenedora

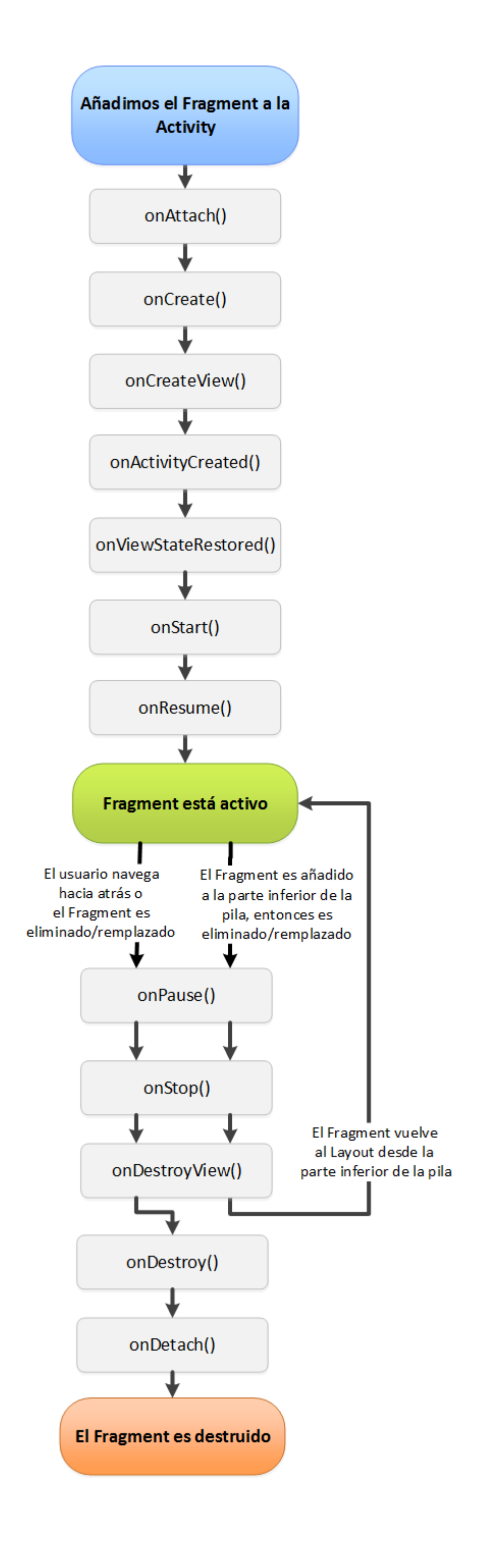
¿Hay que declararlo en el AndroidManifest.xml?

No. Solo se declara su Activity contenedora

Referencias:

- http://jarroba.com/fragments-fragmentos-en-android/
- http://jarroba.com/programar-fragments-fragmentos-en-android/
- http://developer.android.com/guide/components/fragments.html
- http://developer.android.com/reference/android/app/Fragment.html
- http://developer.android.com/training/multiscreen/screensizes.html

Ciclo de vida del Fragment

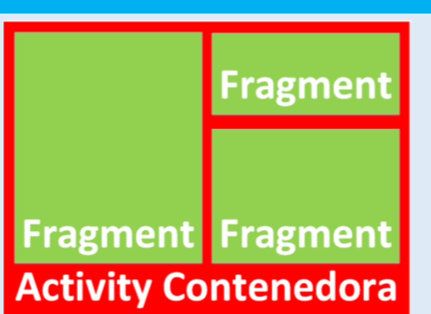


A continuación se muestra en detalle cómo funciona cada método:

Método	¿Cuándo se llama?	¿Qué debería hacer?
onAttach()	Justo después de adjuntar el Fragment a la Activity contenedora	Mantener la instancia de los Callbacks y comprobar si la Activity contenedora los implementa
onCreate() [Uso Recomendado]	Al crearse el Fragment	Aquí deberás inicializar los componentes esenciales del Fragment que quieras conservar cuando sea pausado o parado
onCreateView() [Uso Recomendado]	Llamado en el momento que se necesite pintar la interfaz del usuario por primera vez	Asociar el diseño (un Layout.xml) al Fragment. Para pintar un interfaz de usuario para tu Fragment, tienes que devolver una View en este método que será la raíz del Layout de tu Fragment. Si el Fragment no tiene interfaz gráfica o se devuelve un null o con no sobrescribir este método vale.
onActivityCreated()	Justo después de completarse el onCreate de la Activity contenedora	Por ejemplo, recuperar Views, restaurar estados, retener instancias para los callbacks, etc
onViewStateRestored()	Cuanto se restaura todo el estado que había sido guardado en la jerarquía de las Views del Fragment	Para inicializar el Fragment en función de ciertos datos guardados que un estado
onStart()	Al hacerse visible para el usuario	(Unido al onStar() de la Activity contenedora)
onResume()	Al comenzar la iteración con el usuario	(Unido al onResume() de la Activity contenedora)
onPause() [Uso Recomendado]	Se llama con la primera indicación de que el usuario abandona el Fragment (no siempre significa que el Fragment esté siendo destruido)	(Unido al onPause() de la Activity contenedora)
onStop()	Al no ser visible para el usuario	(Unido al onStop() de la Activity contenedora)
onDestroyView()	Cuando la View que fue previamente creada con onCreateView() sea desajuntada del Fragment	Limpiar las Views asociadas al Fragment (Las referencias que puedan evitar limpiar estas Views)
onDestroy()	Cuando ya no se va a utilizar	Liberar sus recursos y limpiar su estado
onDetach()	Justo antes de que el Fragment deje de estar asociado a su Activity	Cambiar la instancia de los Callbacks a unos vacíos y controlados

¿Qué es una Activity contenedora?

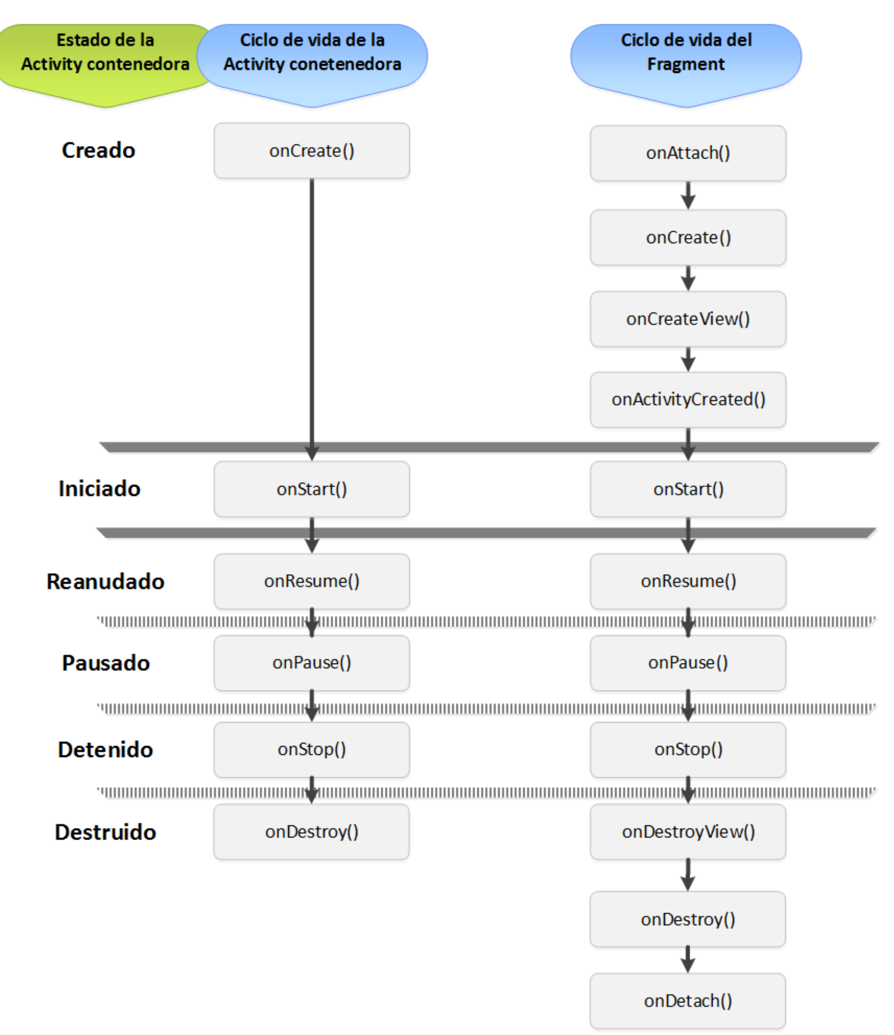
Una Activity contenedora es la Activity que contiene uno o más Fragments. En una aplicación puede haber varias Activities contenedoras. Y un mismo Fragment puede estar sobre varias Activities contenedoras.



Recuerdo que un Fragment siempre tiene que estar sobre un Activity, por lo que todo Fragment tendrá al menos una Activity contenedora.

Cada hueco reservado para un Fragment sobre una Activity contenedora podrá contener a la vez un único Fragment. Si es estático, definido en el diseño de la Activity contenedora como <Fragment/>, no se podrá intercambiar por otro Fragment. Si es dinámico, definido en el diseño por algún ViewGroup, como <FrameLayout/>, desde Java podrá sustituirse un Fragment por otro

Ciclo de vida del Fragment vinculado con el ciclo de vida de la Activity contenedora



Estados del Fragment que coinciden con los de Activity

- **Reanudado** (Resumed): Fragment visible
- **Pausado** (Paused): Otra Activity se ha puesto en primer plano y ha sombreado o se ve parcielmente la Activity contenedora del Fragment
- **Detenido** (Stopped): El Fragment no es visible, pero su estado e información están retenidos por Android. Se puede deber a que la Activity contenedora se haya detenido o que el Fragment se haya eliminado pero añadido a la pila de atrás. El Fragment será matado si lo es la Activity contenedora

Métodos coordinados con el ciclo de vida de la Activity contenedora

- **onAttach**(): cuando el Fragment se asocia a la Activity contenedora
- **onCreateView**(): Al crear la jerarquía de Views asociada al Fragment
- **onActivityCreated**(): al acabar de ejecutarse el método onCreate() de la Activity contenedora
- **onDestroyView**(): cuando se elimina la jerarquía de Views asociada al Fragment
- **onDetach**(): en el momento en que el Fragment se desasocia de la Activity contenedora

Se podrá eliminar y añadir Fragments libremente cuando nos encontremos en el estado de Reanudado de la Activity contenedora

Ejemplo Fragment

Lo que haremos

Crearemos otro nuevo proyecto, pero esta vez será de tipo "**Blank Activity**". Este nos genera la Activity con un Fragment y su diseño asociado. A continuación sustituiremos el código existente por el de este ejemplo.

En la clase Java "MainActivity.java" sustituiremos el código que hay y crearemos otra que la llamaremos "PlaceholderFragment.java" (este nombre no es casual, es el nombre del Fragment que nos crea el ADT con el "HelloWord"; por cuestiones prácticas y de colocación de código preferimos colocarlo en una clase aparte).



Veremos en el LogCat las trazas que se han puesto al arrancar el Fragment, y cuando salgamos de este (pulsando la tecla de atrás).

Nota: Al igual que el anterior ejemplo, aquí solo nos queremos centrar en el ciclo de vida, esta vez de un Fragment.



Ilustración 5 - Aunque esto será lo que veamos en pantalla, no nos interesa en este ejemplo

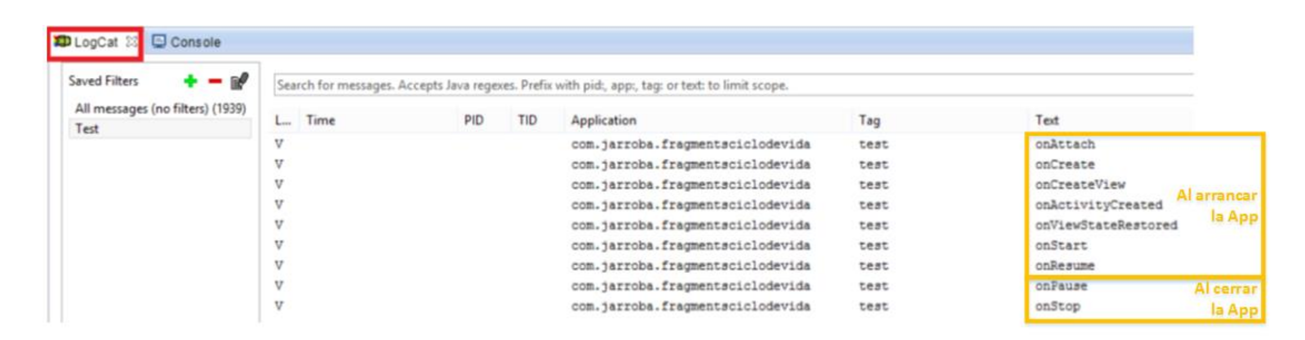
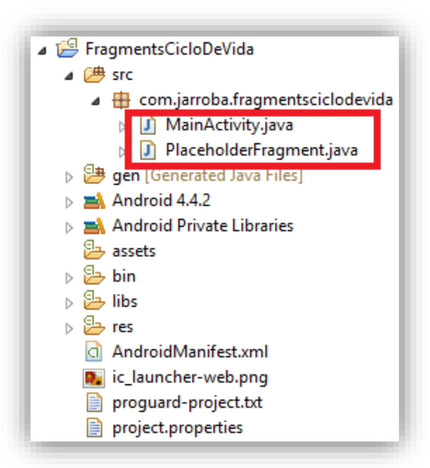


Ilustración 6 - Ejecutaremos la aplicación y al salir de ella nos mostrará en el Log el recorrido del ciclo de vida del Fragment. Aunque es parecido al ciclo de vida de Activity, no es igual, pero sí está emparejado



MainActivity.java

```java
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		if (savedInstanceState == null) {
			getFragmentManager().beginTransaction()
				.add(R.id.container, new PlaceholderFragment())
				.commit();
		}
	}

}
```

PlaceholderFragment.java

```java
import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class PlaceholderFragment extends Fragment {

	private final String LOG_TAG = "test";

	public PlaceholderFragment() {
	}

	@Override
	public void onAttach(Activity activity) {
		super.onAttach(activity);
		Log.v(LOG_TAG, "onAttach");
	}

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		Log.v(LOG_TAG, "onCreate");
	}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_main, container, false);
		Log.v(LOG_TAG, "onCreateView");
		return rootView;
	}
```

```java
        @Override
        public void onActivityCreated(Bundle savedInstanceState) {
                super.onActivityCreated(savedInstanceState);
                Log.v(LOG_TAG, "onActivityCreated");
        }

        @Override
        public void onViewStateRestored(Bundle savedInstanceState) {
                super.onViewStateRestored(savedInstanceState);
                Log.v(LOG_TAG, "onViewStateRestored");
        }

        @Override
        public void onStart() {
                super.onStart();
                Log.v(LOG_TAG, "onStart");
        }

        @Override
        public void onResume() {
                super.onResume();
                Log.v(LOG_TAG, "onResume");
        }

        @Override
        public void onPause() {
                super.onPause();
                Log.v(LOG_TAG, "onPause");
        }

        @Override
        public void onStop() {
                super.onStop();
                Log.v(LOG_TAG, "onStop");
        }

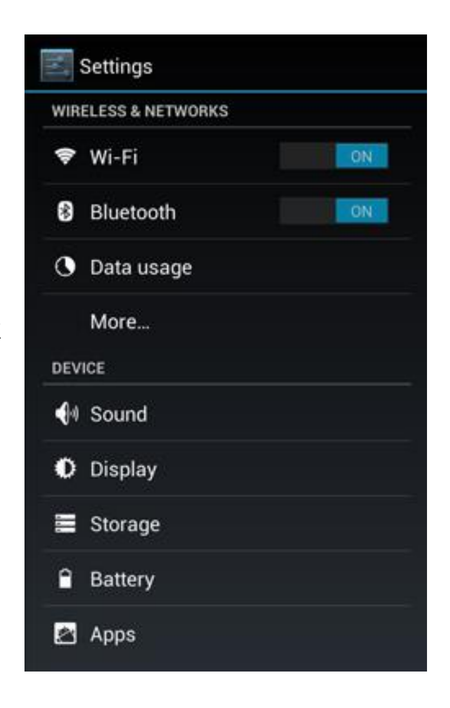
        @Override
        public void onDestroyView() {
                super.onDestroyView();
                Log.v(LOG_TAG, "onDestroyView");
        }

        @Override
        public void onDestroy() {
                super.onDestroy();
                Log.v(LOG_TAG, "onDestroy");
        }

        @Override
        public void onDetach() {
                super.onDetach();
                Log.v(LOG_TAG, "onDetach");
        }

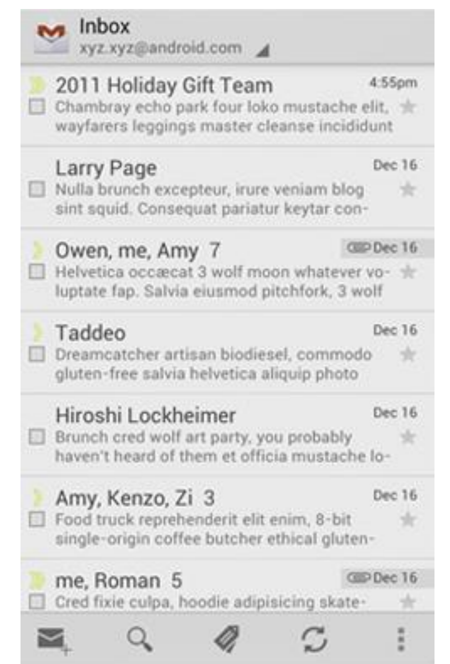
}
```

Diseñar generalmente en Android



¿Hay temas que pueda reutilizar en Android?

Existen dos que podemos reutilizar y modificar a nuestro gusto:

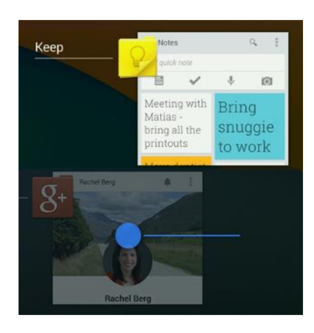


- Holo Light
- Holo Dark

¿Qué Feedback puedo dar al usuario?

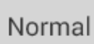


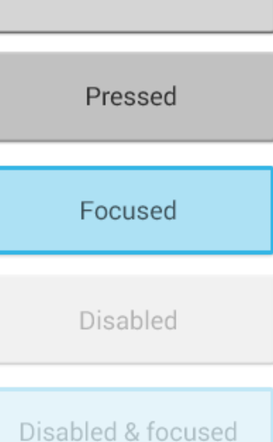
- **Usar el color y la iluminación**: Jugamos con el color u la iluminación para indicar al usuario que ha pulsado o que está pulsado algo.



- **Comunicación:** Jugando con el color, degradados, animaciones y formas, indicaremos al usuario lo que su acción va a provocar (como al empujar con el dedo un elemento de un listado fuera de la pantalla, se transparente para indicar al usuario que se va a eliminar el elemento).

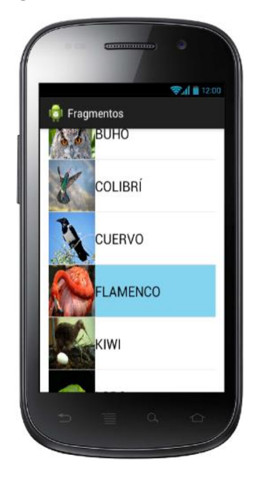
- **Estados**: Definiremos estados (color, forma, sonidos, etc) para indicar al usuario como se encuentra el elemento en ese momento. Por ejemplo, los estados de los botones.



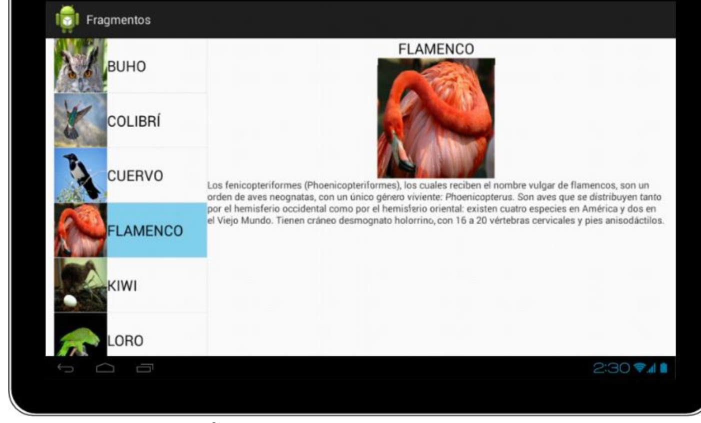


- **Límites**: Indicaremos al usuario que existe un límite (fin o principio de un listado o de escritorio, al tirar se descargarán más datos, etc) con alguna animación o efecto.

¿A qué tendremos que atender con los estilos?



- A la **flexibilidad**, por adaptar los diseños a las diferentes pantallas
- Al **optimizar los diseños**, para aprovechar el tamaño y la forma de las pantallas, variando las vistas para revelar más o menos contenido y facilitar la navegación
- Al proporcionar **recursos para todos**, al disponer de imágenes para todos los tamaños y densidades



Referencias:

- http://developer.android.com/design/style/index.html
- http://developer.android.com/guide/practices/screens_support.html

Métricas y Cuadrículas

¿Cuáles son las unidades de medida en Android? ¿Cuáles se recomiendan utilizar?

Echa un vistazo a la siguiente tabla y lee el siguiente párrafo.

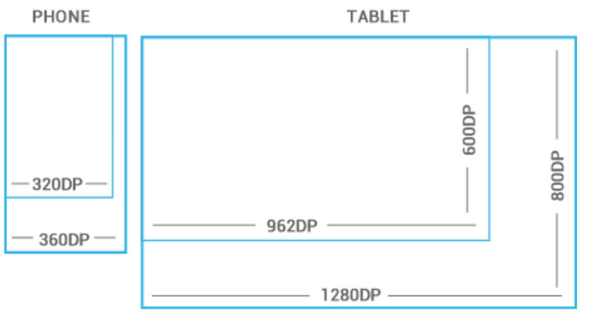
Nombre	Abreviatura	Equivalencia
Píxeles independientes de la densidad (Densisty-independent Pixels)	**dp**	**1 dp = 1 pixel en una pantalla de 160 dpi (mdpi)**
Píxeles independientes de la escala (Scale-independent Pixels)	**sp**	**1 sp = 1 dp * preferencias del tamaño del usuario**
Puntos	pt	1 pt = 1/72 de pulgada
Pixeles	px	1 px = 1 pixel de la pantalla
Milímetros	mm	1 mm = 1 milímetro
Pulgadas (Inches)	in	1 in = 1 pulgada = 25,4 mm

Android soporta píxeles, milímetros, pulgadas y puntos, que están muy bien pero solo serían útiles si solo existiera una única pantalla en todo el mundo. Se puede decir que para los desarrolladores no existen (salvo en casos tremendamente concretos; y con su uso quedarían fuera casi todas las pantallas del mundo al no adaptarse correctamente lo que ve el usuario a los diferentes tipos de pantallas).

Surgió la necesidad de crear los Píxeles independientes de la densidad (dp). Por lo que recomiendo, más bien, verás que es la única manera de que quede bien todos los diseños en todas las pantallas existentes, usar los **dp para las Views.**

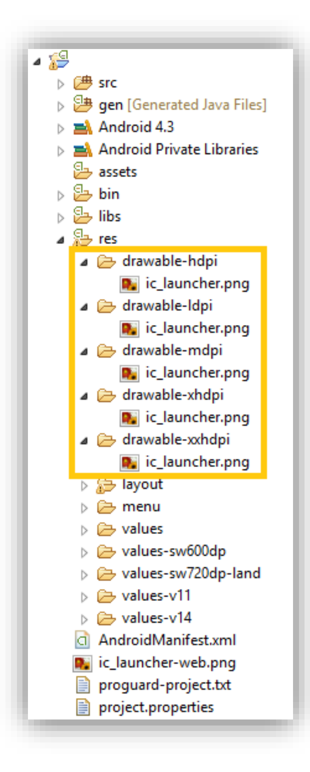
Sin embargo, para los textos se recomienda utilizar Píxeles independientes de la escala (sp). Al utilizar sp para los textos se ofrece al usuario la oportunidad de cambiar el tamaño del texto desde los ajustes del dispositivo. También, se puede decir que es obligatorio utilizarlas, ya que una persona con dificultad en la visión no podría leer nuestra aplicación si utilizamos dp, por ello **para textos utilizaremos sp.**

¿Qué tamaños dan nombre general a los dispositivos?



Veremos más en el tema de Diseño (tema en el que programaremos, este de teoría es para ir teniendo una idea rápida de principios de Android). Pero en términos generales son:

- **Handset** (cabe en una mano, como un SmartPhone, un GPS, etc): igual o menos de 600 dp
- **Tablet** (no cabe en una sola mano): mayor que 600 dp

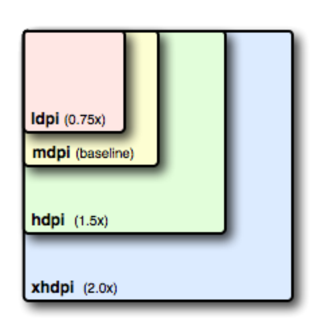


¿Qué densidades existen?

También las veremos más profundamente en el tema de Diseño. En Android las clasificaremos en:

- **XXXHDPI:** densidad alta nivel 4
- **XXHDPI:** densidad alta nivel 3
- **XHDPI:** densidad alta nivel 2
- **HDPI:** densidad alta nivel 1
- **MDPI:** densidad media
- **LDPI:** densidad baja

¿Cómo creo los tamaños y las densidades de las imágenes?



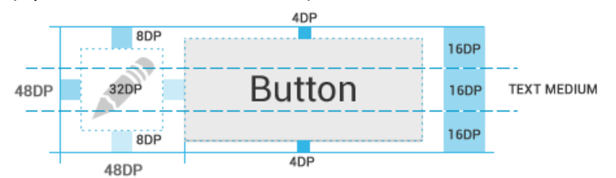
Sencillo, crea una imagen, comprueba que no se aplasta o estira demasiado en una pantalla de 160dpi al utilizarla en la carpeta "mdpi" (que es la medida base) y teniendo ésta de referencia crea las demás.

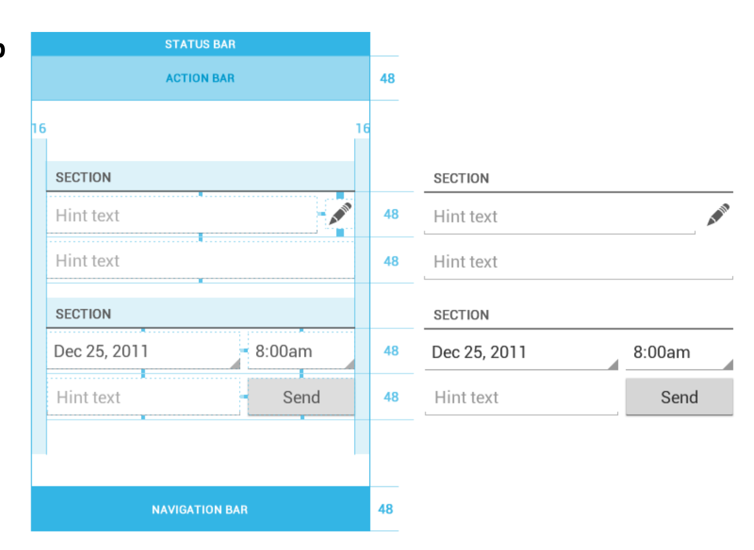
Todas las imágenes con el mismo nombre, irán cada una en una carpeta "drawable" con el sufijo correspondiente (aprende más sobre sufijos en el tema de Recursos).

Nombre	Proporción (ratio de escala)	Densidad	Tamaño de ejemplo del icono "ic_launcher"	Icono de ejemplo de tamaño proporcional
xxxhdpi	4x	640 dpi	192 x 192	
xxhdpi	3x	480 dpi	144 x 144	
xhdpi	2x	320 dpi	96 x 96	
hdpi	1,5x	240 dpi	72 x 72	
mdpi	**1x**	**160 dpi**	**48 x 48**	
ldpi	0,75x	120 dpi	36 x 36	

¿Cuál es la proporción general de todas las Views en Android?

La proporción general de todas las Views será de **48 dp** (aproximadamente 9mm).





Con esta proporción:

- Aseguraremos que las cosas no sean demasiado pequeñas (menos de 7mm), independientemente de la pantalla
- Guardaremos el equilibrio entre la información general, y el objetivo de los elementos de la interfaz de usuario

Tipografía

¿Qué tipografías se incluyen en Android?

La tipografía Roboto. Se puede descargar desde: http://developer.android.com/design/style/typography.html

Roboto
SUNGLASSES
Self-driving robot ice cream truck
Fudgesicles only 25¢
ICE CREAM
Marshmallows & almonds
#9876543210
Music around the block
Summer heat rising up from the sidewalk

¿Qué tipos incluye la tipografía Roboto?

Tipos incluidos: final, ligera, normal y negrita (y cursiva para cada una)

¿Puedo utilizar la tipografía Roboto para otras cosas que no sean Android?

Al contrario que otros elementos de Google que pueden ser más libres, la tipografía Roboto está protegida por una licencia Apache que limita su uso (se encuentra dentro del fichero que contiene la tipografía)

Text Size Micro	12sp
Text Size Small	14sp
Text Size Medium	18sp
Text Size Large	22sp

¿Qué unidad de medida se recomienda utilizar para textos?

El **sp**, como ya se mencionó. Principalmente porque el sistema –previa petición del usuario- puede variar el tamaño de todo lo que declare el formato sp.

Referencias:

- http://developer.android.com/design/style/typography.html

Color

¿Para qué se pude utilizar el color?

Usar el color para dar énfasis

#33B5E5	#AA66CC	#99CC00	#FFBB33	#FF4444
#0099CC	#9933CC	#669900	#FF8800	#CC0000

¿Qué color elegir?

Seleccionar los colores que mejor se ajustan a su marca y ofrecen un buen contraste entre los componentes visuales.

¿Todos los colores se ven bien juntos?

Algunos como el rojo con el azul provocan mareos.

También, indicar que el rojo y el verde no son distinguibles para usuarios daltónicos.

¿Qué notación utilizar para los colores?

El formato RGB (rojo verde azul) es el más recomendado. O su variación más completa el ARGB (alfa rojo verde azul). Siempre en hexadecimal. Se puede usar hexadecimal (como base el número 16, es decir 16 dígitos que van desde el 0 al F) de un dígito (#ARGB) o de dos dígitos (#AARRGGBB).

Es recomendable usar la notación de color **#AARRGGBB**. Cuyo valor mínimo es el #00000000 y el máximo es #FFFFFFFF. Si no se pone AA se considera que está a su máximo valor (FF), que equivale a completamente opaco.

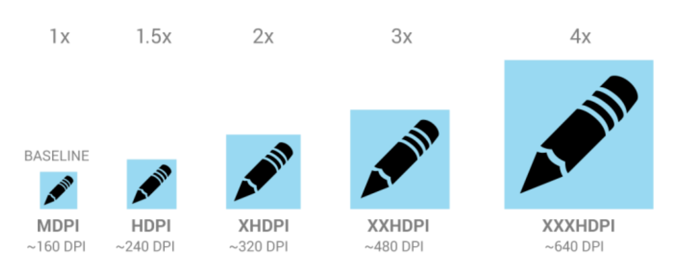
Referencias:

- http://developer.android.com/design/style/color.html

Iconografía

¿Cuántas veces en diferentes densidades tengo que diseñar una imagen para mi aplicación?

A menos que no permitamos que se instale en más de un tipo de pantalla de unas características determinadas (cosa para nada recomendable), tendremos que diseñar más de uno.



Como vimos anteriormente (ver en este mismo tema en “Métricas y cuadrículas” para ver los ratios de escala), las posibilidades en pantalla son infantas, por lo que es más que interesante diseñar un icono para cada una de las densidades de pantalla. Para simplificar será suficiente con diseñar los iconos para las carpetas “drawable” que nos recomienda Android.

Un truco es que no hace falta diseñar todas las densidades, ya que si el tamaño en dpi al dividirlo entre 2 coincide con la escala de una carpeta “drawable”, Android puede escalar la imagen sin perder nada de eficiencia; por ejemplo, un icono en “drawable-hdpi” de 240dpi Android lo escala sin inmutarse a 120dpi, razón por la que no se suele utilizar la carpeta de “drawable-ldpi”. Este truco puede que de la excusa de diseñar un único icono para la carpeta de “drawable-xxxhdpi” y que Android lo escale para el resto, pero esto no es óptimo, Android tardaría un montón de tiempo en escalarla.

¿Cómo se recomienda diseñar los iconos que lanzan la aplicación?

El dibujo se recomienda que sea la representación básica de la aplicación. Por ejemplo, un sobre indicaría que es un gestor de correo.









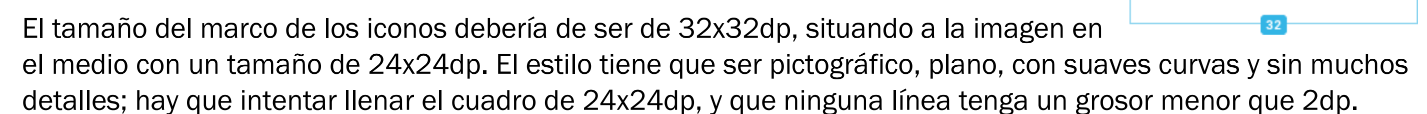
Para dispositivos debe ser de 48x48dp y para la web de Google Play de 512x512 pixeles. Recuerdo que existe la posibilidad de utilizar el asistente de crear un nuevo proyecto Android para que nos genere todos los iconos necesarios de manera automática (todos los necesarios, incluido el de la web Google Play).

El diseño actual sigue una tendencia de iconos tridimensionales con el punto de mira desde un poco por arriba para percibir profundidad. Además, los iconos son redondeados, jugando con sombras y brillos.

¿Cómo diseño los iconos del ActionBar?

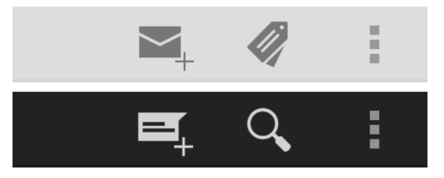
Deben de ser simples metáforas que representen conceptos que la mayoría de la gente pueda entender de un vistazo.

Lo mejor es utilizar los iconos que ya están por defecto en la API de Android, ya que son los símbolos que todo usuario ya ha visto alguna vez. Además, se adaptan a la versión del sistema operativo y a las diferentes pantallas.



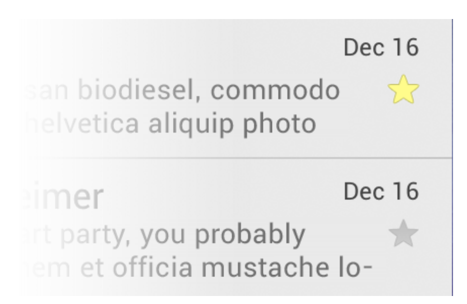
El tamaño del marco de los iconos debería de ser de 32x32dp, situando a la imagen en el medio con un tamaño de 24x24dp. El estilo tiene que ser pictográfico, plano, con suaves curvas y sin muchos detalles; hay que intentar llenar el cuadro de 24x24dp, y que ninguna línea tenga un grosor menor que 2dp.

También hay que estar pendientes de los temas del sistema operativo, ya que tiene dos y hay que diseñar imágenes para ambos:



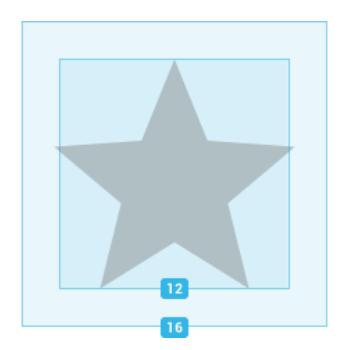
- Holo Light: El color de fondo de #333333, la opacidad cuando está habilitado ha de ser de 60% y para cuando está deshabilitado de 30%
- Holo Dark: El color de fondo de # FFFFFF, la opacidad cuando está habilitado ha de ser de 80% y para cuando está deshabilitado de 30%

¿Cómo diseño los iconos acción, de estado y de menú contextual?

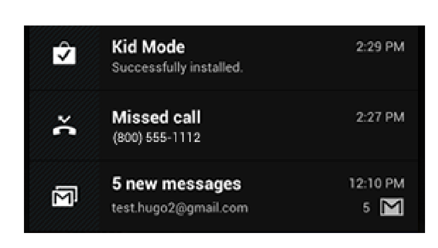


Por ejemplo el icono de marcar como favorito.

El tamaño del marco que sea de 16x16dp, y el de la imagen central de 12x12dp. De estilos parecidos a los de la ActionBar; de estilo natural, plano y simple, de colores que contrasten, eligiendo las metáforas visuales que mejor entienda y recuerde el usuario.

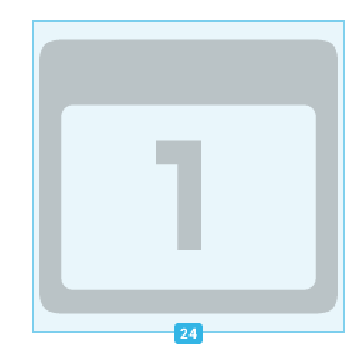


¿Cómo diseño los de las notificaciones?



Un icono que aparecerá en la notificación en la barra de estado (la barra que tiene el reloj y la batería del móvil).

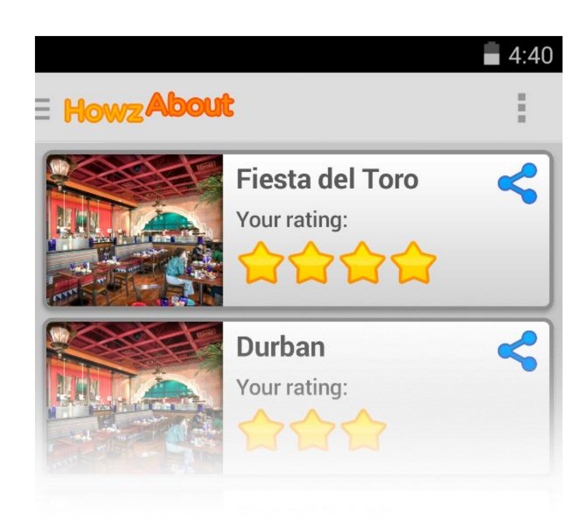
El tamaño del marco debería de ser de 24x24dp, siendo el dibujo de 22x22dp. Y el estilo que sea liso y simple, utilizando la misma metáfora que el icono que lanza la aplicación (para indicar al usuario que esa notificación es de tu aplicación), y de color blanco.



Referencias:

- http://developer.android.com/design/style/iconography.html

Diseña tu propia marca



¿Tienen que ser todas las aplicaciones iguales?

Seguir la guía de estilo de Android no significa que todas las aplicaciones se tengan que ver iguales

¿Qué puedo cambiar?

Casi todos los estilos e imágenes como: Logo, nombre y colores de la Action Bar, fondo, estilo de texto, etc

¿Tienen que ser los iconos de los botones obligatoriamente iguales?

No, hay unos recomendados, pero se puede aplicar un diseño propio. De este modo se pueden hacer iconos que se ajusten con nuestra marca o estilo.

Lo único que hay que cumplir, que estos iconos tengan la misma representación de la metáfora de Android. Por ejemplo, si queremos diseñar el icono de compartir que sean los 3 puntos unidos, no una flecha u otra metáfora.





Referencias:

- http://developer.android.com/design/style/branding.html

Consejos para los textos

¿Qué consejos me puedes dar para escribir los textos de una aplicación?

- **Brevedad:** Conciso, simple y preciso. Se recomienda que el límite sean 30 caracteres
- **Sencillez:** Palabras cortas, verbos sencillos y nombres comunes. Es decir, escribir para gente que no entienda bien el idioma o se estén iniciando en el mismo
- **Amigable:** Usar contracciones. Hable con cortesía y directamente al lector en segunda persona del singular (tutear o formalmente con tratamiento de usted). Que el texto transmita al usuario seguridad, alegría y energía
- **Poner primero lo más importante:** Las primeras 2 palabras deberían incluir una muestra de la información más importante
- **Describir solo lo necesario:** No tratar de describir sutilezas, pues pierden a los usuarios
- **Evitar la repetición:** Que no se repitan los términos, aunque estén escritos con diferentes palabras, más de una vez

Ejemplo de texto mal escrito	¿Por qué está mal el ejemplo de la izquierda?	Ejemplo de texto mejor escrito
Consulte la documentación que viene con su teléfono para disponer de instrucciones adicionales	Demasiado formal y largo	Lea las instrucciones que vienen con su teléfono
Pulse siguiente para completar la instalación usando una conexión Wi-Fi	Por estar la información importante en último lugar	Para terminar la instalación usando Wi-Fi, pulse siguiente

Referencias:

- http://developer.android.com/design/style/writing.html

Carpeta "res"

¿Dónde se encuentran los recursos?

Todos los recursos se encuentran en la carpeta "res" (que viene de las tres primeras letras de la palabra recursos en inglés "resources"), clasificados en carpetas con un nombre específico y posiblemente con sufijos.

¿Qué puede contener la carpeta "res"?

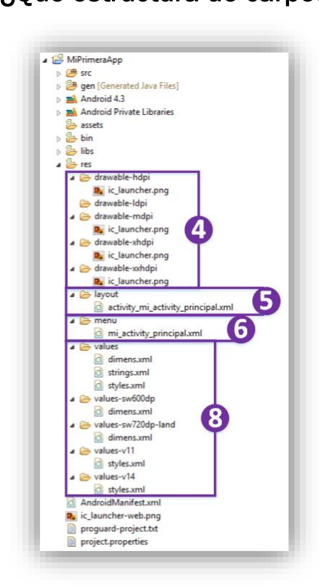
La carpeta de recursos contendrá las imágenes como iconos o logos, los textos en uno o varios idiomas, las dimensiones, los estilos, colores, animaciones, entre otras cosas que sean siempre finales y que no vayan a cambiar a lo largo del programa.

Todo lo introducido en esta carpeta será optimizado automáticamente. Esto quiere decir que, por ejemplo, una imagen posiblemente no tendrá el tamaño, formato o resolución original.

¿Puedo poner cualquier nombre a las carpetas?

No se puede poner cualquier nombre a las carpetas dentro de "res", han de seguir una estructura

¿Qué estructura de carpetas sigue la carpeta "res"?



La estructura interna de esta carpeta es (iremos usándolas con ejemplos según avancemos en contenido en el libro):

1. **animator**: XML con propiedades de las animaciones (ejemplo: tiempo de interpolación)
2. **anim**: XML con los estados visuales de las animaciones (ejemplo: que el objeto animado empiece el pequeño y acabe grande)
3. **color**: XML con la lista de colores en #RGB, #ARGB, #RRGGBB o #AARRGGBB
4. **drawable**: imágenes (.png, .9.png, .jpg, .gif), o XML que definan imágenes
5. **Layout**: XML con la interfaz de usuario
6. **menu**: XML con los menús
7. **raw**: cualquier tipo de fichero (ejemplo: audios)
8. **values**: XML múltiples de valores simples (ejemplos: textos, enteros, arrays, etc). Se han estandarizado: arrays.xml, colors.xml, dimens.xml, strings.xml, styles.xml)
9. **xml**: XML arbitrarios que pueden ser leídos en tiempo de ejecución

¿Puedo poner cualquier nombre a los ficheros de los recursos?

No, están definidos y no se pueden poner otros nombres.

Todos los ficheros de recurso han de ir en minúscula, sin caracteres extraños (salvo guion bajo “_” o dólar “$”) y sin espacios. Ejemplo: “mi_imagen.png” o “mi_fichero.xml”

Dicho de otro modo, los nombres de los recursos deben ser escritos como si escribiéramos variables en Java (pero todo en minúsculas) seguido de la extensión (“.xml”, “.png”, etc). Esto es así, porque sus nombres van a ser recogidas por el fichero “R.java” y convertidas en variables.

¿Cómo añado un recurso que NO sea XML?

Por ejemplo, puedes añadir imágenes en formato PNG para crear los iconos de la aplicación. Es tan sencillo como arrastrar el recurso a la carpeta correspondiente. Para una imagen que tengamos en el escritorio de nuestro ordenador, la arrastraremos desde el escritorio a dentro de Eclipse, hasta la carpeta de prefijo “drawable” que vayamos a necesitar.

¿Cómo se crea un nuevo recurso XML?

Si creamos un nuevo recurso XML para un tipo de configuración de dispositivo determinado, el ADT nos generará automáticamente la carpeta (si no existe) para guardar ahí el recurso.

Se hace de una manera muy sencilla. Podemos pulsar en la barra de tareas el icono de una “a” con un símbolo de suma en amarillo.

O podemos pulsar con el botón derecho del ratón sobre la carpeta del proyecto (o sobre la carpeta “res” también vale, por si lo ves más natural, aunque da igual) en el que queramos crear un nuevo recurso e ir a “New/Android XML File”.

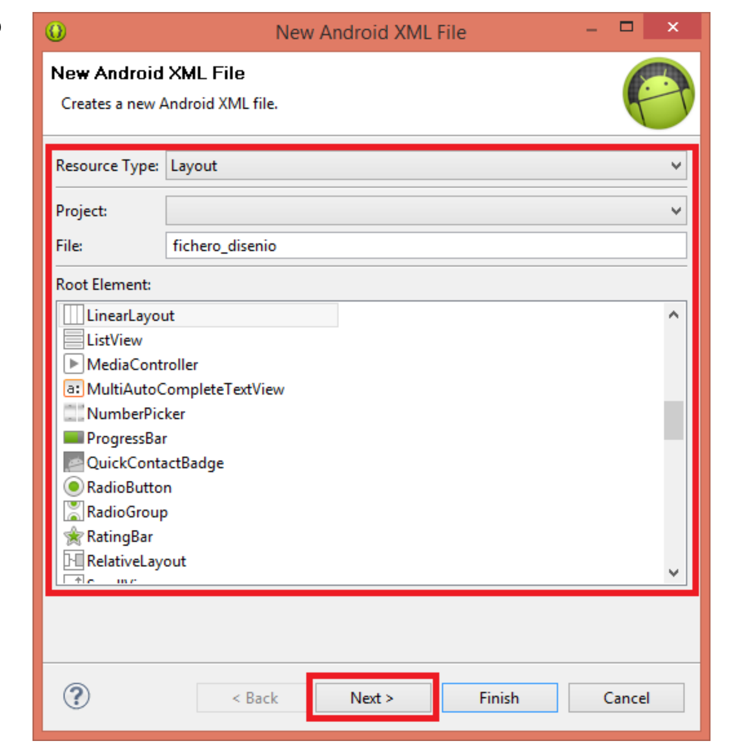


En la ventana que se abre llamada "New Android XML File" definiremos el tipo de recurso XML que crearemos con:

- **Resource Type** (Tipo de recurso): elegiremos el tipo de recurso que queramos crear, existen varios. Estos son:

Recurso	Traducción	Carpeta	Descripción
Layout	Diseño	layout	Un fichero de diseño de qué es lo que verá el usuario en la pantalla
Values	Valores	values	Valores varios como estilos, dimensiones, strings, arrays, etc.
Drawable	Dibujables	drawable	Cualquier imagen que definamos mediante XML
Menu	Menú	menu	Elementos de los menús o de la barra de acciones (ActionBar)
Color List	Lista de colores	color	Para guardar toda nuestra paleta de colores en hexadecimal
Property Animation	Propiedad de la animación	animator	Define propiedades de una animación. Como las veces que se tiene que repetir una animación, la duración, etc
Tween Animation	Interpolación de la animación	anim	Define las interpolaciones de las animaciones, como si un objeto tiene que rotar, moverse de un punto a otro, etc
AppWidget Provider	Proveedor del AppWidget	xml	Cuando creemos un AppWidget de escritorio tendremos que declarar aquí sus propiedades, tales como el tiempo de refresco, el diseño que tendrá, etc
Preference	Preferencias	xml	Para crear los Activities (PreferenceActivity) o Fragments (PreferenceFragment) de preferencias de usuario
Searchable	Buscable	xml	Para en los campos de búsqueda activar el autocompletar, entre otras cosas

- **Project** (Proyecto): Seguramente ya esté elegido si lo hemos seleccionado previamente.
- **File** (Fichero): Nombre del fichero que crearemos (no hace falta ponerle la extensión XML).
- **Root Element** (Elemento Raíz): El elemento que definirá la raíz de nuestro fichero XML. Dependerá del tipo de recurso que hayamos seleccionado.



Por ejemplo, en la imagen adyacente crearemos un fichero de diseño para nuestro proyecto, que llamaremos "fichero_disenio" y que su elemento raíz sea un <LinearLayout>.

Pulsaremos el botón de "Next >" que nos conducirá a una nueva ventana. Pero antes la explicaremos.

¿Cómo funciona lo de los sufijos en las carpetas de recursos?

Las carpetas de recurso pueden tener sufijos que indicarán a Android que recursos usar en cada momento. Momentos como la orientación del dispositivo, el idioma, el tamaño de la pantalla, la versión de Android, etc. Por ejemplo: drawable-hdpi (hdpi indica que son imágenes para pantallas de alta resolución) o "values-sw720dp-land" ("sw720dp" indica que esa carpeta está dirigida a pantallas de más de 720x1280dp, es decir, tablets de 10 pulgadas; y "land" indica que solo para cuando está la pantalla en horizontal).

Creamos una nueva carpeta dentro de la carpeta "res". En el campo de nombre de la carpeta la llamamos "layout-" seguido de un sufijo de los anteriores. Por ejemplo, podemos crear: "layout-land", "layout- sw123dp", "layout-large", entre otras. Veremos cómo se hace esto en las siguientes preguntas.

Existen ciertos sufijos que se pueden combinar, como por ejemplo "layout-large-land".

Veremos una explicación más profunda sobre sufijos en carpetas en el tema de "Diseños". Aunque funciona para todas las carpetas de recursos de manera similar.

¿Cuáles son los sufijos para las carpetas de recursos?

Existen varias maneras de clasificar los recursos, unos más nuevos que otros. Aunque funcionan todos, se recomienda utilizar los nuevos.

- **Sufijos para diferentes pantallas:**
  - **Sufijos antiguos** (obsoletos): Por ejemplo "layout-large-land", indicará que solo tomará los recursos de esta carpeta si el dispositivo donde se ejecuta tiene una pantalla grande, y se encuentra en posición horizontal.

Tipo	Sufijo	Recursos de diseño para:
Tamaño	small	Pantallas pequeñas (obsoleto)
	normal	Pantallas normales
	large	Pantallas grandes
	xlarge	Pantallas muy grandes
Densidad	ldpi	Densidad baja, unos 120 dpi
	mdpi	Densidad media, unos 160 dpi. Es la base de todas las medidas.
	hdpi	Densidad alta, unos 320 dpi
	xhdpi	Densidad muy alta
	xxhdpi	Densidad mucho más alta
	nodpi	Todas las densidades. Android no modificará el tamaño de estos recursos.
	tvdpi	Densidad media-alta, unos 213 dpi (obsoleto)
Orientación	land	Posición horizontal o apaisado (landscape)
	port	Posición vertical (portrait)
Relación de aspecto (Aspect ratio)	long	Pantallas muy largas
	notlong	Pantallas de aspecto similar al de base (el tipo de pantallas mayoritario)

- **Sufijos nuevos** (cubren todas las posibilidades de los sufijos antiguos): Por ejemplo, si queremos que solo utilicen unos recursos las Tablets, usaremos "layout-w820dp".

Nombre	Fromación del sufijo	Sufijo de ejemplo	Recursos de diseño para:
Ancho más pequeño	sw + (nº entero) + dp	sw123dp	Anchuras de pantalla que no sean menores al tamaño indicado, independiente de la orientación del dispositivo; es decir, que si el dispositivo está en posición horizontal, se tomará como anchura la altura (aquí se interpreta la anchura como la longitud más estrecha). Esta es la alternativa a los tamaños anteriores (small, normal, large, xlarge)
Anchura de pantalla disponible	w + (nº entero) + dp	w456dp	Mínimo de anchura de la pantalla. Aquí la anchura es siempre la distancia entre el extremo de izquierda y el de derecha de la pantalla; por lo que cambiará al girar el dispositivo.
Altura de pantalla disponible	h + (nº entero) + dp	h789dp	Mínimo de altura de la pantalla. Aquí la altura es siempre la distancia entre el extremo superior y el inferior de la pantalla; por lo que cambiará al girar el dispositivo.

Indicar que los sufijos antiguos se pueden combinar, como por ejemplo "layout-large-land". Los nuevos no se pueden combinar, ya que no es necesario.

- **Sufijos de idioma**: Por ejemplo, tendremos recursos de idiomas en español en la carpeta "values-es", o en francés en la carpeta "values-fr". Recomiendo poner el inglés en la carpeta "values" sin sufijo (de este modo, un dispositivo cuyo no esté recogido en las carpetas con sufijo, entrará en ésta).

Idioma	Sufijo
(Idioma por defecto)	(Si no ponemos sufijo, se tomará por defecto esta carpeta de no encontrarse alguna del idioma adecuado. Recomiendo que el idioma por defecto sea el inglés)
Inglés	en
Español	es
Portugués	pt
Indio (Hindi)	hi
Árabe	ar
Chino	zh
Francés	fr
Alemán	de
Italiano	it
Japonés	ja
Coreano	ko

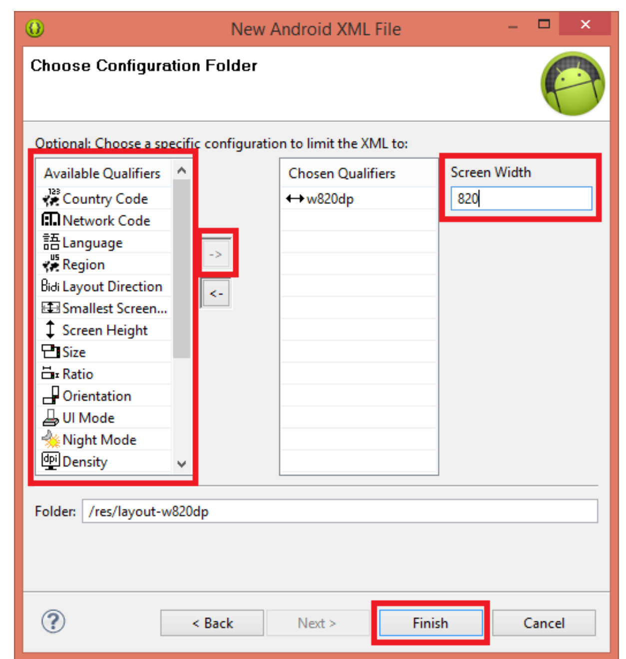
- **Sufijos de versión de Android**: Por ejemplo, si creamos una carpeta llamada v14 significará que solo entrará, si la aplicación se ejecuta en un dispositivo que tenga como mínimo la versión 4.0 de Android.

Nombre Android	Versión Android	API	Sufijo correspondiente
Cupcake	1.5	3	v3
Donut	1.6	4	v4
Eclair	2.0/2.1	7	v7
Froyo	2.2	8	v8
Gingerbread	2.3.2/2.3.7	9/10	v9/v10
Honeycomb	3.0/3.1/3.2	11/12/13	v11/v12/v13
Ice Cream Sandwich	4.0/4.0.3	14/15	v14/v15
Jelly Bean	4.1.2/4.2.2/4.3	16/17/18	v16/v17/v18
KitKat	4.4/4.4.3	19	v19

¿Cómo creamos una nueva carpeta de recurso, y si queremos con un sufijo, de manera automática?

Aunque se puede crear a mano y añadir el sufijo. Debido a que es imposible saberse todos, en este libro vamos a recomendar un truco que -pese a no estar muy extendido- es muy cómodo, útil y no querrás dejar de usarlo.

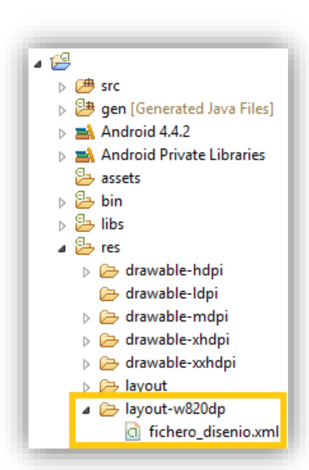
Continuaremos desde la ventana que dejamos antes.



Pulsaremos “Next >” para ir al siguiente paso donde elegiremos el tipo de configuración de la carpeta de recursos (Si no existe se creará). Es decir, en qué carpeta con qué prefijo se guardará el fichero.

En este ejemplo, estamos creando un fichero de diseño que queremos que solo funcione para un tipo de pantalla de ancho mínimo de 820dp, para que solo sirva para Tablets.

Por lo que elegiremos en el panel de la izquierda “Avaible Qualifiers” (Elementos que cualifican disponibles), el elemento “Screen Width”. Y pulsaremos la flecha de mandarlo a la derecha, a la columna “Chosen Qualifiers” (Elementos que cualifican elegidos). Aquí podremos especificar sus propiedades; en el caso de este ejemplo, pondremos 820 para indicar la anchura mínima en dp.

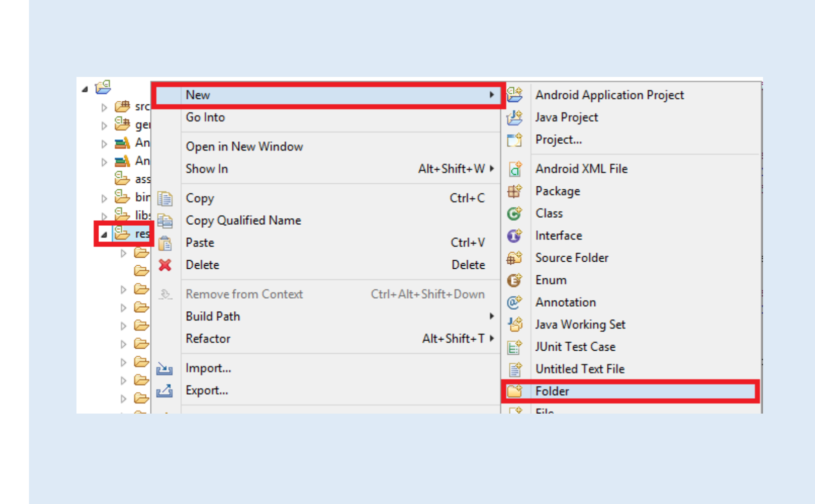


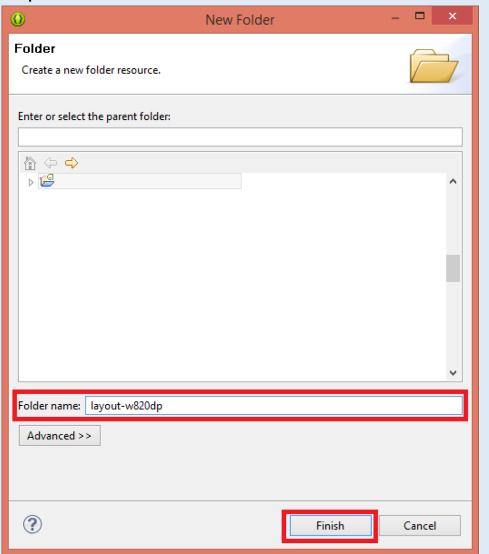
Ya podremos pulsar el botón de “Finish”.

Descubriremos en la estructura de nuestro proyecto Android, como efectivamente se nos habrá creado la carpeta con el sufijo y dentro nuestro fichero de diseño XML. En el ejemplo que hemos explicado veremos la carpeta “layout-w820dp” con el fichero que creamos “fichero_disenio.xml”.

¿Cómo se crea una nueva carpeta de recurso de manera manula?

Aunque recomiendo la manera automática, podría ser útil en ciertas ocasiones.
Para **crear una nueva carpeta en Eclipse** para guardar nuestros recursos: elegir la carpeta “**res**” **con el botón derecho del ratón**. Ahí pulsar en elegimos “**New**” y luego “**Folder**”. Se abrirá una nueva ventana en al que pondremos el nombre de la nueva carpeta de recursos, por ejemplo “layout-w820dp”.
En la imagen de ejemplo se muestran las carpetas que se generan de manera automática al crear un proyecto nuevo. El resto que se mencionan aquí, si se necesitan utilizar habrá que crearlas a mano.



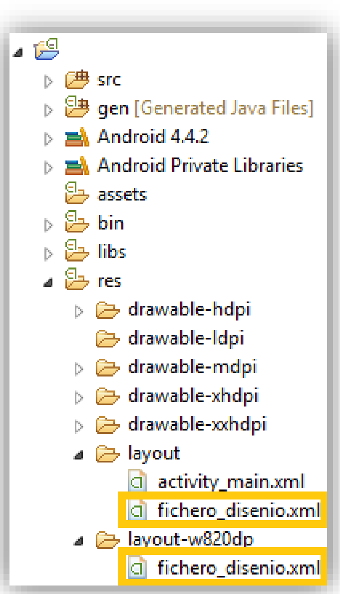


¿Qué ocurre si la configuración del dispositivo encaja entre dos carpetas de recursos?

Android utilizará la carpeta más cercana que esté por debajo. Si son imágenes, los recursos de esta carpeta, los estirará hasta completar le tamaño que necesita la pantalla.

Por ejemplo, supongamos que tenemos las carpetas de recursos: "layout", "layout-w320dp" y "layout-w820dp". Si la pantalla de nuestro dispositivo es de 480dp, Android nos devolverá los recursos de "layout-w320dp". Por otro lado, si la pantalla fuera inferior a 320dp entrará en la carpeta por defecto "layout". Con las versiones de Android ocurre lo mismo. Y con los idiomas, se devuelve la carpeta "values-XX" del idioma del dispositivo; o si no existiera ninguna, directamente "values".

¿Podremos tener varios ficheros de recurso con el mismo nombre?



Sí, pero en carpetas diferentes. Supongamos que queremos hacer un diseño para Tablets y otro para el resto de dispositivos (como los Smartphones). Crearemos el mismo nombre de fichero las carpetas correspondientes y así Android tomará el que precise, dependiendo en cuál dispositivo se ejecute la aplicación.

Android primero buscará en la carpeta que corresponda. Por ejemplo, si queremos cargar en una Activity el "fichero_disenio.xml", y la aplicación se ejecuta en un Tablet, primero buscará en la carpeta "layout-w820dp", como lo encuentra carga este. Por otro lado, el fichero "activity_main.xml", si se ejecuta la aplicación en un Tablet, no lo encontraría en la carpeta "layout-w820dp", por lo que tomará el de la carpeta "layout". Si la aplicación se ejecuta sobre un Smartphone, directamente se buscará en la carpeta "layout"; pues no cumple la condición de la carpeta "layout-w820dp", al no llegar la pantalla del Smartphone 820dp de anchura.

Evidentemente los dos ficheros "fichero_disenio.xml" serían diferentes, sino no tendría sentido crear dos ficheros, pudiendo meter todo junto en la carpeta "layout".

Referencias:

- http://developer.android.com/guide/topics/resources/providing-resources.html
- http://developer.android.com/training/basics/supporting-devices/languages.html
- http://developer.android.com/training/basics/supporting-devices/screens.html
- http://www.ethnologue.com/statistics/size
- http://developer.android.com/training/basics/supporting-devices/languages.html
- http://developer.android.com/reference/java/util/Locale.html
- http://es.wikipedia.org/wiki/ISO_639-1
- http://www.loc.gov/standards/iso639-2/php/code_list.php

Context

¿Es necesario entender cómo funciona el Context?

Si estás empezando a programar en Android, al principio no es necesario que conozcas todos sus detalles, así que este tema si quieres échale una ojeada rápida y ve al siguiente.

Digo esto porque el Context es de esas cosas que se entienden cuando ya llevas alguna aplicación hecha en Android (Es como ¿qué va antes si el huevo o la gallina? ¿Aprender Context antes o después de hacer aplicaciones?). En Android se utiliza mucho el Context, y la mejor manera de entenderlo es practicándolo, por lo que lo entenderás "durante", ni antes ni después. Eso sí, cuando ya te hayas pegado con el Context unas cuantas veces te recomiendo que vuelvas a leerte este tema, ya verás cómo muchas cosas serán reveladas con una segunda lectura después de la práctica.

Si ya tienes un cierto nivel en Android, entender cómo funciona el Context en profundidad será de vital importancia para programar con soltura en Android.

¿Qué es?

Es el interfaz global de información acerca del entorno de la aplicación. Es decir, nos proporciona a los programadores acceso a métodos que nos facilitan la vida. Como lanzar nuevas Activities con startActivity() (¿A qué no programamos cada línea de código de lanzar una aplicación? Pues no, ya que lo hace el Context), obtener recursos de la carpeta res con getResources() o los Strings con getString(), obtener los datos almacenados en local con getSharedPreferences(), entre otros muchos.

El contexto es justo eso, un contexto. Proporciona un acceso a todo el entorno de la aplicación; es decir, permite el acceso a los recursos de la aplicación (como a las imágenes, colores, etc) o del sistema operativo para Activities, Services u otros componentes. Además, cada Context está ligado al ciclo de vida del elemento que cubre.

Es una clase abstracta que implementa Android.

Los elementos que tienen Context heredan de Context. Otros que no tienen Context, como View, necesitan que se lo pasen para acceder a los recursos (imaginemos una Activity que tiene un tema oscuro, y otra que tiene un tema claro; si le pasamos el Context de la Activity de tema claro a la View, cargará un tema claro y no el oscuro).

¿El contexto es siempre "this"?

No, es parte pero no el total. No confundamos "this" que la llamada al objeto mismo (para llamar a sus variables globales y métodos del propio objeto). Para poder llamar a "this" la clase tiene que heredar de una clase que herede de Context.

Además, no siempre se llama con "this". Existen varios tipos de Context, y se obtendrán de diferentes maneras dependiendo desde donde lo queramos obtener y qué tipo queramos obtener. Esto lo respondemos en las siguientes preguntas.

¿Cómo se obtiene?

Se puede obtener desde (iremos viendo ejemplos a medida que avancemos por el libro):

- Una clase que extiende de Context (como Activity, Service, Application, etc): **this**
- Una clase que extiende de Context desde una clase anónima: **ClaseActivity.this**
- Una clase que NO extiende de Context (como View): View.**getContext()**
- La aplicación: Activity.**getApplicationContext()**
- Otro context: ContextWrapper.**getBaseContext()**
- La Activity del Fragment que lo contiene: **getActivity()**

Nota: ContextWrapper actúa de proxy entre contextos, delegando las llamadas de un contexto a otro

¿Qué tipos de existen?

Existe el Context de:

- **Aplicación**: Como es lógico, al cubrir todo el ciclo de vida de la aplicación desde que la arrancamos hasta que muere, cada aplicación tiene un único contexto de aplicación; además, este contexto engloba a todos los demás. Lo que hace Android es crear una instancia Singleton que se crea al iniciar el proceso de la aplicación. Se puede acceder desde una Activity o un Service con getApplication(), o desde cualquiera que herede de Context con getApplicationContext().
- **Activity o Service:** Heredan de ContextWrapper que a su vez hereda de Context. ContextWrapper hace que se apodere de todas las llamadas y las envíe a una instancia oculta de Context (conocido como "Base Context")

¿Todos los componentes de Android heredan de Context?

No, estos dos no heredan de Context (por lo que Android se los pasa de algún modo):

- **BroadcastReceiver**: Pero se le pasa un Context en el onReceive() cada vez que un nuevo evento difundido por el sistema entra.
- **ContentProvider**: Puede acceder al Context con getContext() y se le devolverá el Context de la aplicación que lo esté ejecutando (puede ser la misma aplicación u otra diferente).

¿Quién suele pedir el Context?

Normalmente piden el Context (iremos viendo ejemplos a medida que avancemos por el libro):

- **Al acceder a los recursos** como imágenes, Strings, etc
- **Al crear nuevos Views, Listeners, Adapters, Intents, etc.**
- **Al acceder directamente a componentes** como ActionBar, Intents, etc.

¿Es seguro utilizar el Context en todos los casos?

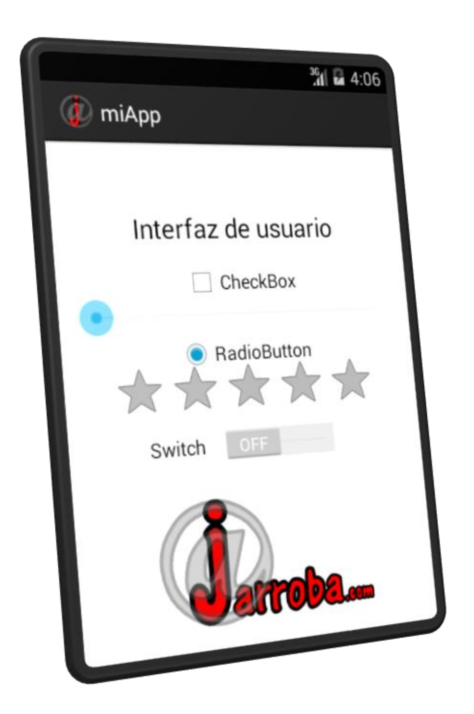
A veces puede producir fugas de memoria que es mejor evitar. Lo mejor y ante la duda de si se provocará una fuga de memoria o no suele ser mejor utilizar: getApplicationContext().

Aunque si comprendes perfectamente la utilización del Context, es aconsejable utilizar el Context apropiado en cada ocasión.

Referencias:

- http://jarroba.com/context-de-android/
- http://developer.android.com/reference/android/content/Context.html

Layout



¿Qué es?

Un diseño (Layout) define una estructura visual. Es la interfaz gráfica de la aplicación.

¿Quién puede mostrar un diseño gráfico?

Normalmente lo van a mostrar Activity, Fragments o App Widget.

¿Dónde se declara un diseño?

O bien, estáticamente en la carpeta "res/layout" en XML, que es lo recomendable por consumir menos recursos, al tardar menos en procesarse; o en Java en tiempo de ejecución, que dependiendo de lo que tenga que procesar, podría llegar a bloquear al usuario.

¿Cuál es la mejor manera de declararlo?

En XML, porque separa la capa de representación del código que controla el comportamiento. Además, es más fácil encontrar los errores, ver la estructura, y se puede contemplar el resultado final en tiempo real mientras el desarrollador crear la interfaz.

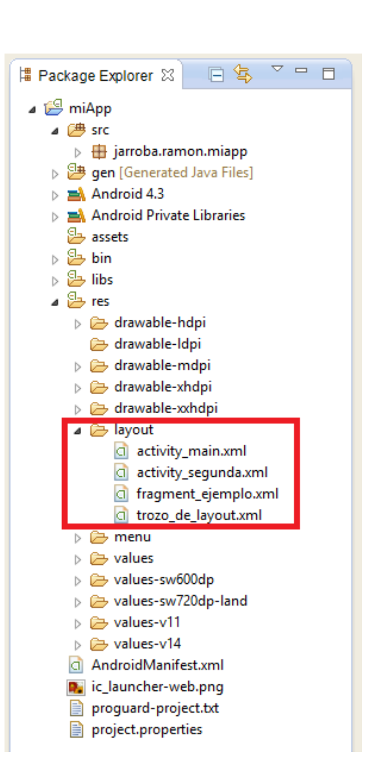
¿No hay que declarar ningún DTD en los XML?

Es un XML pero no hay que declarar ningún DTD (El "<?Doctype … >" no hay que ponerlo), debido a que Android usa etiquetas dinámicas

Referencias:

- http://developer.android.com/guide/topics/ui/declaring-layout.html

Carpeta Layout



¿Cantidad de Views que puede mantener directamente?

Una

¿Qué es una View?

Un elemento para construir la interfaz de usuario (lo veremos con detalle más adelante)

¿De qué actúa?

De raíz del árbol de Views

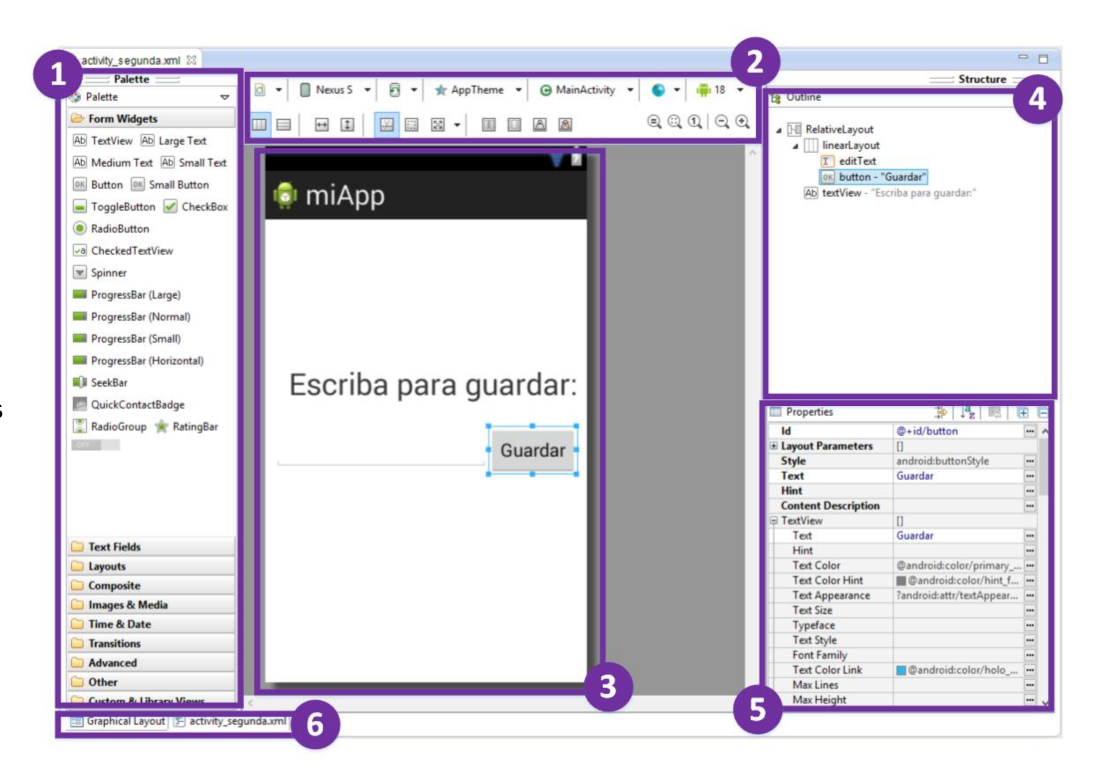
¿Qué tipo de elemento puede mantener como raíz del fichero?

ViewGroup, View o <merge>

Entorno gráfico para diseñar la Interfaz de usuario

Para diseñar una Interfaz tenemos que dirigirnos al directorio res/layout. Ahí crearemos o editaremos ficheros XML.

Editar un fichero de diseño de interfaces gráficas



Abriremos el editor de diseños gráficos haciendo doble clic sobre un fichero de diseño (Un fichero XML de la carpeta "layout").

Este editor consta de las siguientes partes:

1. **Palette**: Donde las Views se pueden arrastrar al Canvas o al Outline. De aquí podremos arrastrar cualquier elemento (estos elementos son los denominadas Views) tanto al Canvas como al Outline.
2. **Configuration Chooser**: Barra con las diferentes configuraciones de pantalla y de renderizado para la View que seleccionemos (se explican a continuación)

3. **Canvas**: Editor visual. Aquí veremos en tiempo real como va quedar el diseño. Una sugerencia, si tienes varias Views ya colocadas, puede que sea misión imposible seleccionar con el ratón la que queramos aquí, por lo que recomiendo usar el Outline para mayor comodidad a la hora de seleccionar una View

4. **Outline**: Jerarquía en árbol de las Views. Muy práctico para trabajar con las diferentes capas que componen la interfaz gráfica (aunque no es siempre así, son capas como las de los programas de edición de imágenes; dicho de otro modo, si tenemos dos Views, una encima de otra, la que aquí aparezca más arriba cubrirá a la de más abajo) y su jerarquía

5. **Properties**: Propiedades de la View seleccionada. Donde podremos editar su texto, color, tamaño, etc (también se explicará en detalle más adelante)

6. **Pestañas para cambiar entre el "Diseñador gráfico" y el "código XML"** que se genera. Por motivos prácticos, en este libro ofrece el código XML, que habrá que copiar en la pestaña de XML para seguir los tutoriales. Y en ciertas ocasiones convendrá trabajar directamente en esta pestaña XML, pues el diseñador gráfico no es tan bueno como gustaría

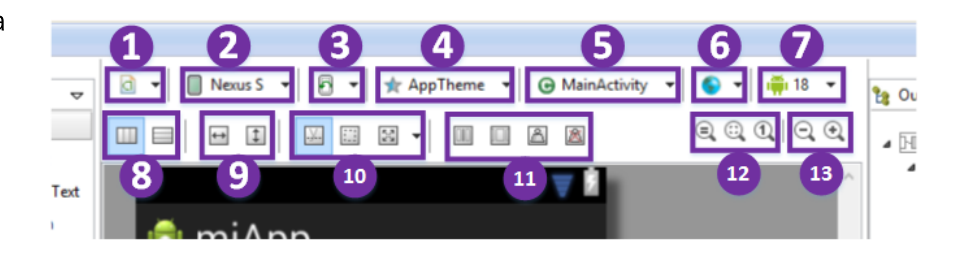
Referencias:

- http://developer.android.com/tools/help/adt.html

Configuration Chooser

Este menú de configuración podrá variar algún botón dependiendo de la View seleccionada y en dentro de que otra View está. Se va a explicar el más genérico. Esta descripción corresponde a la captura anterior, donde hemos seleccionado un Button dentro de un LinearLayout:

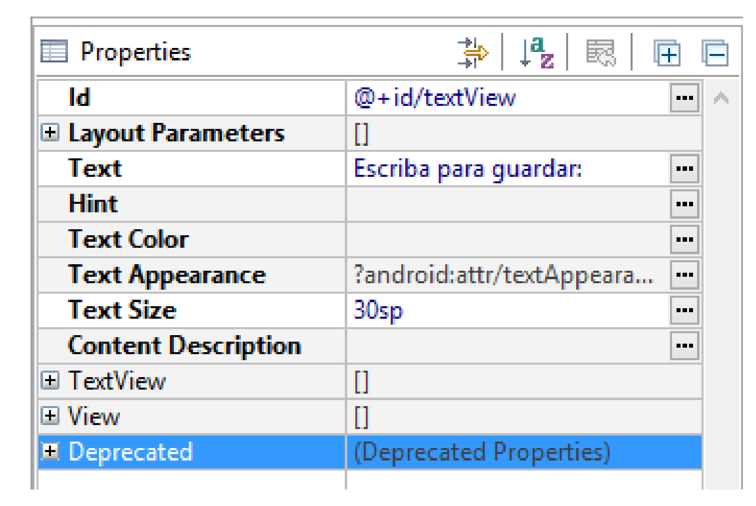
1. Añadir **previsualizaciones** en miniatura
2. Cambiar el **tamaño** de pantalla
3. **Rotar** pantalla
4. **Tema**
5. **Actividad que asocia al Layout**
6. Cadenas de **texto**
7. **Vista previa del diseño de las Views dependiendo de la versión** del sistema operativo (esto es solo una vista previa, que nos muestra cómo quedará según qué versión, no obliga a ello)
8. **Posición** vertical u horizontal de las Vies dentro de un ViewGroup
9. **Rellenar** a lo ancho o alto
10. **Alinear** a la línea base, **Márgenes** y **Gravedad**
11. Distribuir los **pesos** por igual a todos los elementos, Asignar todo el peso al elemento seleccionado, Poner un peso, Eliminar todos los pesos
12. **Emular un tamaño** real de los píxeles, Hacer zoom hasta rellenar el Layout en la pantalla, Hacer zoom hasta el 100%
13. Aumentar o disminuir el **zoom**



Properties

Existen varias agrupaciones de las propiedades:

- **Nivel superior**: Los atributos que se recomiendan rellenar siempre (estarán dentro de otros niveles, cambiado aquí o en el nivel correspondiente cambiará al otro directamente, pues es el mismo atributo)
- **Layout Parameters**: Parámetros respecto al padre (un GroupView) y variarán según el padre. Usado para posicionar el cuadro contenedor de la View. Ejemplo: Centrar la View respeto al padre
- **(Nombre de la View o View Padre)**: parámetros especiales para la View. Ejemplo: Para un TextView será el contenido del texto
- **View**: parámetros comunes para todas la Views. Ejemplo: El fondo o el estilo
- **ViewGroup**: parámetros solo de las ViewGroup. Ejemplo: Animar a los hijos al modificarlos
- **Deprecated**: parámetros anticuados que no se deben usar



Propiedades de las Views



¿Cómo es la geometría de todas las View?

Rectangular

¿De qué se compone una View en el diseñador gráfico?

- **Marco del área de la View.** Es un rectángulo invisible que delimita el área total que ocupa la View. Este rectángulo puede estar ajustado o no a su contenido. Es invisible para el usuario, pero el desarrollador lo puede ver en el diseñador gráfico al seleccionar una View.
- **Uno o varios dibujos que mostrar al usuario.** Pueden tener forma de texto, botón, cuadro seleccionable, una imagen cualquiera, etc. Las llamadas ViewGroup no tienen un dibujo propio, sino que al contener otras Views -que tienen rectángulo que marca su área y dibujo- tendrán sus dibujos.

¿Las propiedades afectan al dibujo o el rectángulo que la contiene?

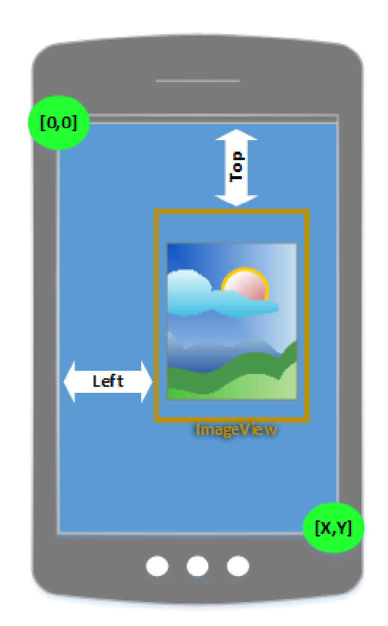
Normalmente las que están contenidas en el recuadro Properties en el apartado "nombre de la View en cuestión" afectan sólo al contenido; como puede ser cambiar el color al texto. El resto afectarán a las dos, es decir, si afectan al rectángulo que contiene al dibujo, entonces afectará al dibujo; por ejemplo, si hacemos desplazamos el rectángulo del área, el dibujo se moverá junto a éste.

¿Qué diferencia hay entre las propiedades a la misma View y respecto al padre (Layout Parameters)? ¿A caso el padre no tiene sus propias propiedades?

No hay que confundir entre las "propiedades **del** padre" con las "propiedades **respecto** al padre". Las primeras son las que le afectan directamente a la View que es padre de alguna otra View, como cambiar el fondo (Background). Y las que son respecto al padre definen por ejemplo la colocación respecto a este (especificando, por ejemplo, que la View hija está 5 píxeles a la izquierda respecto al marco del padre), o decidir si ajustar el tamaño de la View a todo lo que nos deje el padre (dicho de otro modo, agrandar la View hasta que choca su marco con el de la View padre).

Nota: A continuación explicaremos algunas de las propiedades más complejas, otras son muy sencillas de entender y no merecen más explicación (como Background que es fondo, o Text que es el texto, etc).

Posición (Position)



¿Cuál es el pixel que posiciona una View?

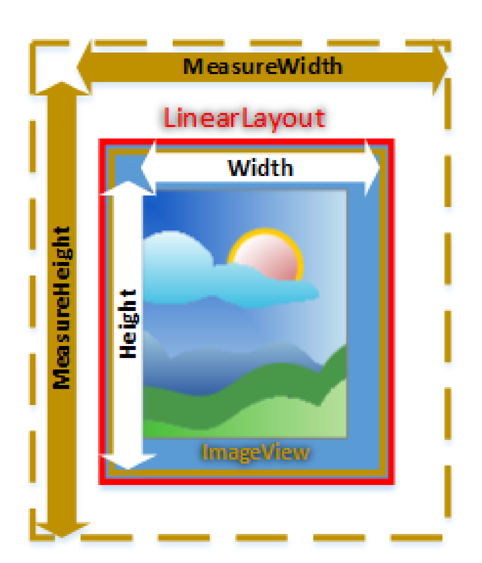
En la coordenada [0,0], que corresponde con la esquina superior izquierda de la View. Las Views se colocan siempre en relación **left** (a la izquierda o coordenada X) y **top** (la parte superior o coordenada Y)

¿Cómo se obtiene en Java la posición?

Cuando se obtiene la posición con **getLeft()** o **getTop()** se obtiene en píxeles, y es siempre respecto a la esquina superior izquierda de la pantalla (coordenada [0,0])

Para facilitar las cosas, se ofrece también **getBottom()**, que es lo mismo que hacer getTop()+getHeight(); y **getRight()**, que equivale a hacer getLeft()+getWidth().

Tamaño (Size)



¿Cómo se define el tamaño?

El tamaño de una View se define con **width** (anchura) y **height** (altura).

¿Cómo se obtiene en Java la anchura medida y altura medida?

Las dimensiones medidas se obtienen en píxeles con **getMeasuredWidth()** y **getMeasuredHeight()**. Definen cuanto de grande quiere ser la View dentro del padre.

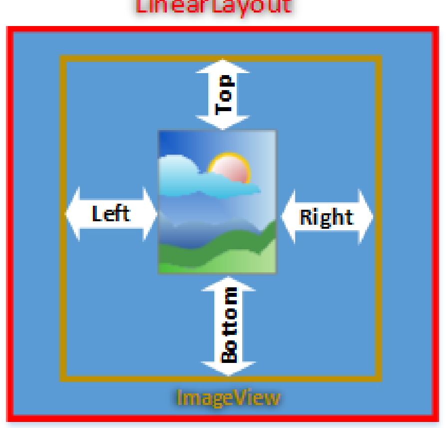
En la imagen de ejemplo, podemos ver que la ImageView quiere ser más grande que el LinearLayout (supongamos que el LinearLayout mide 100x200 y que contiene una ImageView que quiere medir 150x250)

¿Cómo se obtiene en Java la anchura dibujada y altura dibujada?

Se obtiene en píxeles con **getWidth()** y **getHeight()**. Definen el tamaño actual que tiene la View en la pantalla. No deberían de ser diferentes las dimensiones dibujadas que las medidas (que siempre coincidan recae sobre el desarrollador).

En la imagen de ejemplo, podemos ver que la ImageView tiene el tamaño el LinearLayout (en el ejemplo anterior indicamos que el LinearLayout mide 100x200 y la ImageView quería medir 150x250; pero como el padre, que es el LinearLayout, no le permite crecer, la ImageView solo conseguirá como máximo el tamaño del padre, es decir, un tamaño de 100x200).

Relleno (Padding)



¿Qué es?

Rodea a una View (left, top, right, bottom). Se utiliza para compensar el contenido de la View; es decir, para que el contenido del rectángulo de la View se desplace en su interior

¿Añadir Padding cuenta para el tamaño de la View?

Sí, ya que modifica la forma del rectángulo que define a la View

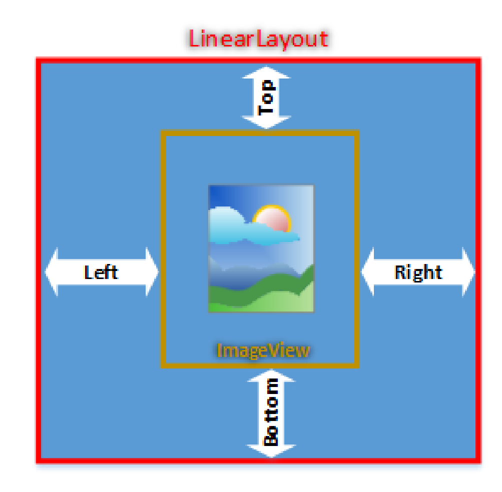
¿El Padding pertenece al padre o a la misma View?

El Padding pertenece a la View donde se define

Margen (Margin)

¿Qué es?

Desplaza a una View (left, top, right, bottom). Se utiliza para posicionar con más precisión a la View



¿Añadir Margin cuenta para el tamaño de la View?

No, ya que solo la desplaza

¿El Margin pertenece al padre o a la misma View?

El Margin pertenece al padre de la View que lo define

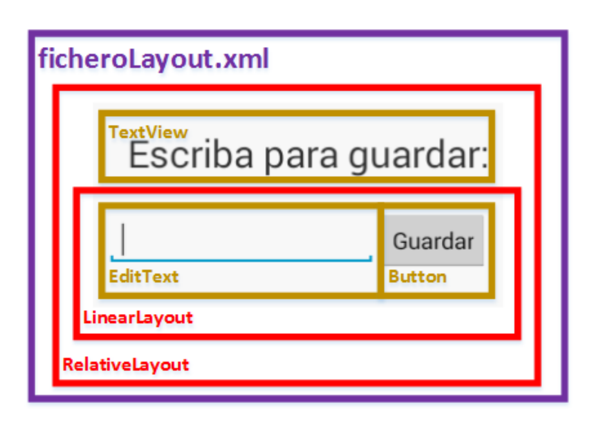
Ejemplo de crear una diseño (Layout) o vista en XML desde cero

Lo que haremos

Queremos conseguir el siguiente diseño. Para ello modificaremos el diseño que ya existen en la carpeta "layout" que se llama "fragment_main.xml". Así quedará guardado en un fichero XML, que se encuentra en la carpeta "layout" que a su vez está dentro de la carpeta "res".

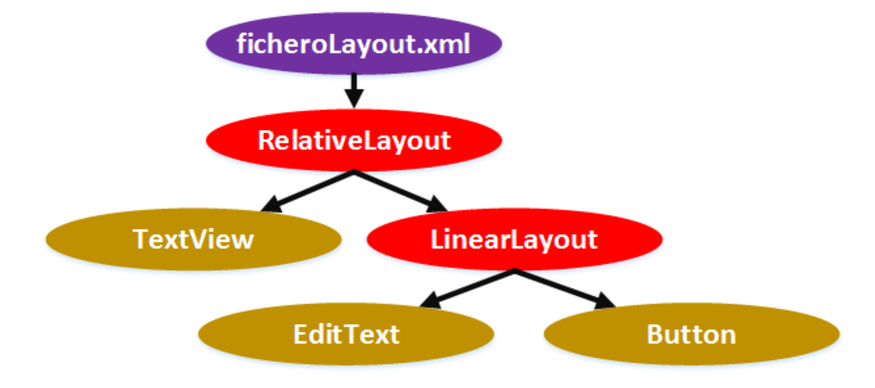


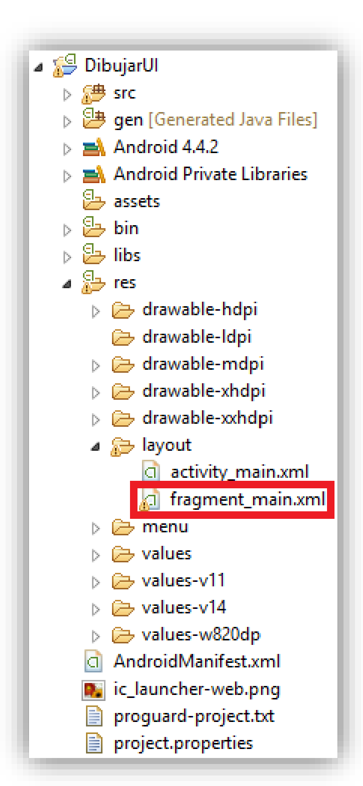
Utilizaremos tanto Views como ViewsGroup. Deshaciendo mentalmente el diseño en los siguientes elementos:



La raíz (sin contar el fichero, que no se suele contar) será un **RelativeLayout** para poder colocar donde queramos en pantalla los elementos **TextView** y un **LinearLayout**. El LinearLayout lo colocaremos en formato horizontal para que dé cabida a la View **EditText** y al **Button**; además, de agrupándolos en un solo grupo para cuando los tengamos que mover nos resulte más cómodo.

Nos dará el siguiente árbol mental (estos pasos son obligatorios si tenemos un diseño previo que queremos hacer realidad, las primeras veces que toquemos la herramienta recomiendo jugar con el diseñador gráfico).



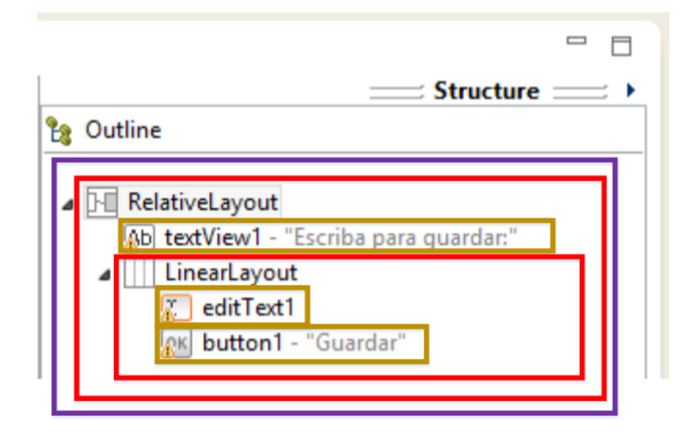


¿No utilizamos el fichero de diseño "activity_main.xml"?

En este ejemplo no lo utilizaremos. El fichero "activity_main.xml" se encarga de dar un diseño a la Activity, pero ya tiene código que ayudará a que contenga al fichero "fragment_main.xml". Podrás deducir que una Activity puede contener Fragments. Recuerdo que el código que genera automáticamente el SDK de Android en Eclipse, monta un Fragment de ejemplo dentro de una Activity. Ya que tenemos el Fragment trabajaremos sobre él (es decir, se deriva el grueso de la Vista al Fragment; antiguamente se hacía todo directamente en el Activity, aquí daremos lo nuevo y veremos que se hace igual). Lo veremos más adelante, aquí nos centraremos en diseño.

Lo primero que haremos será crear otro proyecto básico, como ya hicimos. Iremos a la ruta "res/layout" y abriremos el archivo llamado "fragment_main.xml".

Crearemos la estructura de la interfaz gráfica arrastrando elementos del panel de "Palette". Hasta que cumpla lo que queremos tanto en el panel de "Canvas" (el cómo lo verá el usuario) y en el "Outline" (el cómo quedará la estructura de árbol que queríamos):



¿Aparecen símbolos amarillos con exclamación y código subrayado de amarillo?

Únicamente son advertencias (Warnings), que habrá que corregir, pero que permite ejecutar y que funcione nuestra aplicación perfectamente. Nos preocuparemos si sale subrayado en rojo o con círculos rojos con una X blanca. Explicaremos como corregir estas advertencias; con ello, aprenderemos a hacer bien las cosas. De momento lo dejaremos así.

Tocaremos algunas propiedades para ajustar algunas cosas, como los textos y que quede clavado a lo que queremos.

Luego cambiaremos a la pestaña de "Código XML" y veremos el código generado (también podríamos haberlo diseñado desde aquí; y créeme, en ciertas ocasiones es mucho más cómodo)

res/layout/fragment_main.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.jarroba.dibujarui.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="130dp"
        android:text="Escriba para guardar:"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="31dp" >

        <EditText
            android:id="@+id/editText1"
            android:layout_width="188dp"
            android:layout_height="wrap_content"
            android:hint="">
            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Guardar" />

    </LinearLayout>

</RelativeLayout>
```

Podemos comprobar que este código también cumple perfectamente lo que diseñamos desde un principio:



¿Ejecutamos este código?

Podemos ejecutarlo en el emulador o en un dispositivo, como ya se enseñó. Pero de momento solo estamos diseñando las Vistas. Todavía no hemos dado funcionalidad. No te impacientes, todo llegará antes que tarde :)

Elementos de diseño XML

Nombre y apertura de la View
```
<LinearLayout
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center|top"
    android:orientation="vertical" >
</LinearLayout>
```
Atributo — Nombre — Valor — Cierre de la View — View

Después del primer vistazo de cómo se forma un diseño (Layout) en Android, pasamos a explicar para qué sirven los elementos y sus partes.

Estos elementos XML se componen de una etiqueta de apertura **<**start-tag**>** y cierre **</**end-tag**>** (el ejemplo de la derecha es de estos)**,** o directamente con etiquetas vacías **<**empty-tag**/>**. Contendrán atributos opcionales, mayormente su nombre comenzará con el espacio de nombres “android:” y luego la propiedad a la que afectan, como por ejemplo “android:text”. Cada elemento tiene sus propiedades.

Los elementos son:

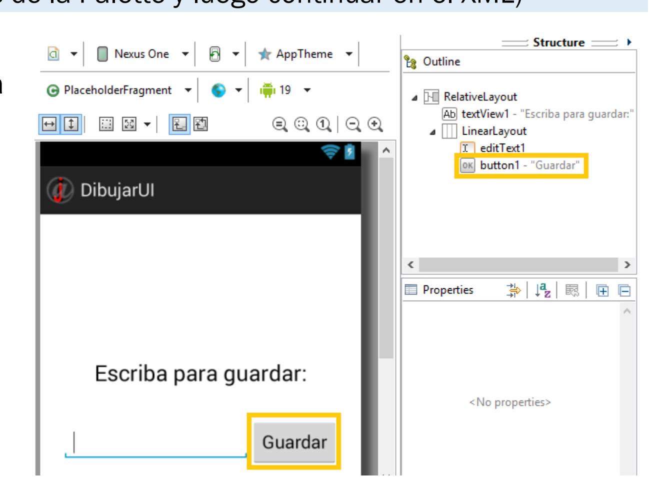
¿Si hay editor que nos hace el código para qué aprender el código XML?

Aunque puedas ver esta parte como un extra para aprender todos los entresijos de Android; la verdad es que el editor todavía es un poco carente, y con algunos problemas que se solucionan al escribir directamente en XML. Esto no quita que usemos el editor, ya que también nos ayudará a colocar las cosas y a crear código más deprisa en algunas partes (como para crear las Views con los atributos más usados, es decir, arrastrar las Views de la Palette y luego continuar en el XML)

View

¿Qué es?

Un bloque básico para construir interfaces de usuario. Dicho de otro modo, un componente individual de la interfaz gráfica



¿Cómo se declara en XML?

Con el nombre de la **<View/>**

- Si son **individuales**: se escriben con etiquetas vacías (Salvo con RequestFocus, ver más adelante). Un ejemplo es: <Button/>
- Si son **grupales**: se escriben con etiquetas de apertura y cierre**.** Un ejemplo es: <LinearLayout></LinearLayout> (más en ViewGroup)

```
<Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Guardar" />
```

¿Cuánto ocupa?

Un área rectangular de la pantalla

¿De qué se encarga?

De dibujar el elemento y de controlar los eventos de éste

¿Cantidad de Views puede mantener directamente?

Ninguna

¿Cómo se disponen las diferentes Views?

En forma de árbol, cuya raíz es el fichero Layout

¿Se le conoce por otro nombre?

Widget (no confundir con las “App Widget” que se pueden colocar en el escritorio de Android)

¿Cuáles son estas Views?

- **Individuales:** TextView, EditText, ProgressBar, ImageView, Button, CheckBox, etc
- **Grupales**: LinearLayout, FrameLayout, etc (más en ViewGroup)

Referencias:

- http://developer.android.com/reference/android/view/View.html

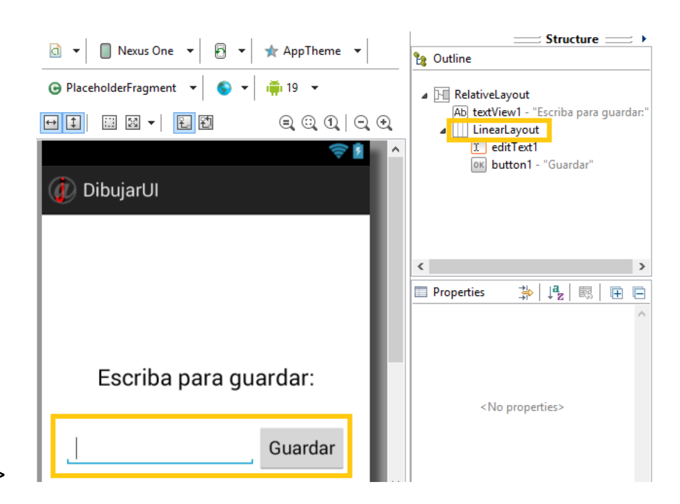
ViewGroup

¿Qué es?

Es una View que sirve de contenedor de Views

¿Cómo se declara en XML?

Con el nombre de la **<ViewGroup></ViewGroup>**. Se escriben con etiquetas de apertura **<**start-tag**>** y cierre **</**end-tag**>**, debido a que contendrán otras etiquetas de otras Views. Un ejemplo es: <LinearLayout></LinearLayout>



¿Cantidad de Views puede mantener directamente?

Normalmente las que queramos, algunas solo una para funciones específicas. Todas la Views que estén dentro de esta ViewGroup son denominadas hijas

```
<LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="31dp" >

        <EditText
            android:id="@+id/editText1"
            android:layout_width="188dp"
            android:layout_height="wrap_content"
            android:hint="">
            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Guardar" />

</LinearLayout>
```

¿No hemos dicho qué una View no puede contener a ninguna otra?

Son la excepción. Aunque hereda de View y por tanto es una View, está diseñado para contener varias Views

¿Es bueno usar muchas ViewGroup anidados?

No, cuanto menos se usen mejor. Además, cuanto más superficial sea el árbol de Views mejor; cuanto más profundo sea el árbol más tarda en dibujarse en pantalla. Es decir, un árbol ancho es mejor que uno profundo

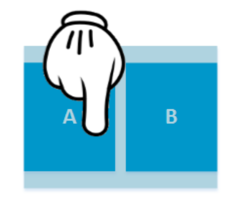
¿Cuáles son estos ViewGroups?

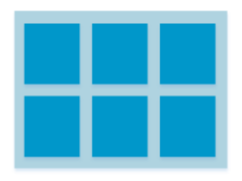
- **LinearLayout**: Organiza a sus hijos en filas horizontales o en verticales. Además, crea una barra de Scroll si la longitud es superior al tamaño de la pantalla

- **FrameLayout:** Reserva el espacio en pantalla para un único hijo (Normalmente este hijo será un Fragment). Para organizar Views hijos de una manera que posibilite el escalado para los diferentes tamaños de pantalla, sin que se sobrepongan unos a otros. Se pueden añadir varios hijos al FrameLayout; de este modo, si se le añaden más de un hijo, se irán apilando y solo se mostrará el último en añadirse.

- **RelativeLayout**: Especifica la colocación de los hijos en relación a la colocación de los otros hijos (Ejemplo: el Button a la derecha del EditText), o respecto a sus bordes

- **ViewPager:** Permite pasar de una página a otra con el dedo a derecha o izquierda. Dichas páginas son conjuntos de Views.





- **GridLayout:** Organiza sus hijos en una cuadrícula

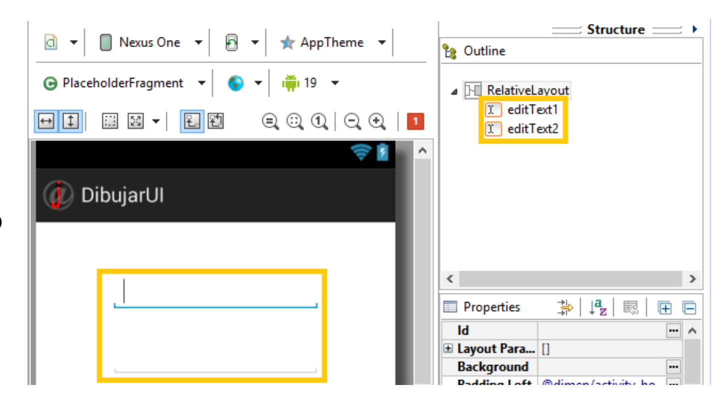
- **WebView:** Muestra páginas web



Referencias:

- http://developer.android.com/reference/android/view/ViewGroup.html

RequestFocus



¿Qué es?

Cualquier View puede incluir a este elemento vacío. Al iniciarse la pantalla, proporciona el foco al padre que lo contiene

¿Cómo se declara en XML?

Con el nombre de la **<RequestFocus/>**. Se escribe con etiquetas vacías **<empty-tag/>** dentro de la View que queramos que tenga el foco.

```
<EditText
        android:id="@+id/editText1"
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:ems="10" >
        <requestFocus />
</EditText>

<EditText
        android:id="@+id/editText2"
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:ems="10" />
```

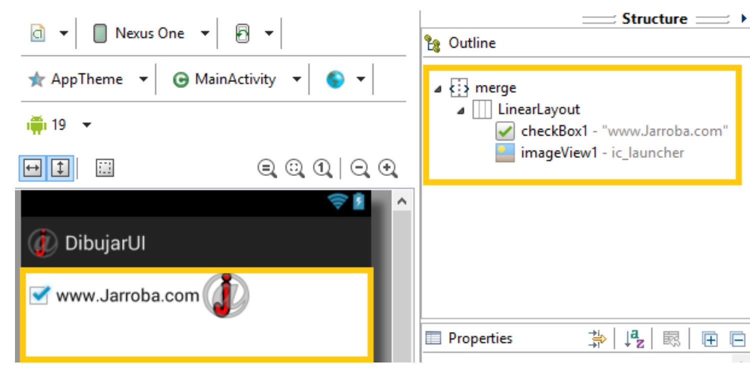
¿Cuántos tags RequestFocus puede haber por pantalla?

Solo se pude haber uno de estos por fichero de diseño. Ya que solo un elemento puede tener el foco a la vez, y en el estado de inicio –que es como se declara en el diseñador gráfico- solo se le puede asignar a una View.

Merge

¿Qué es?

Un elemento raíz que no se debe dibujar en la jerarquía de los Layouts



¿Cómo se declara en XML?

Con **<merge></merge>**

¿Realmente sirve para algo?

Muy útil cuando el diseño de un fichero vaya a ser incluido mediante un <include> (ver Include) en otro fichero de diseño, y este fichero no quiera abusar de profundidad de un ViewGroup que no le vaya a servir de nada

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center" >

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center" >

        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true"
            android:text="www.Jarroba.com" />

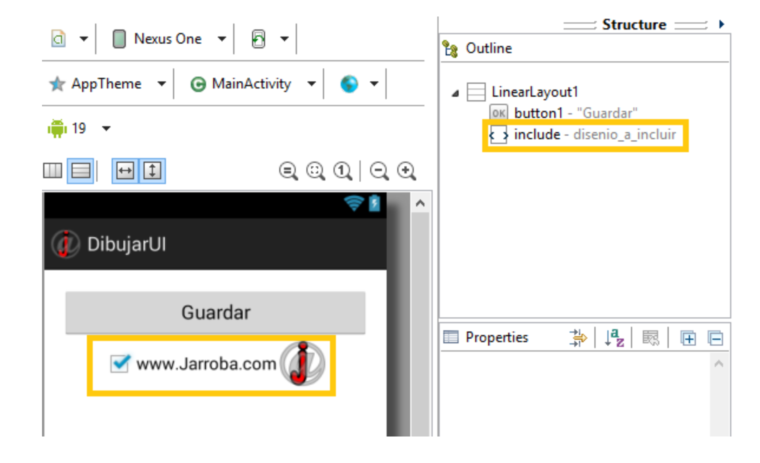
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
    </LinearLayout>

</merge>
```

Include

¿Qué es?

Incluye un fichero de diseño XML dentro del que lo declara



¿Cómo se declara en XML?

Con el nombre **<include/>**. El fichero se declara en el atributo "**layout**" buscándolo con: **@layout/nombre_fichero_layout** (si ponemos la View de include desde la "Palette" el asistente nos pondrá los atributos necesarios, nosotros solo tendremos que elegir el fichero que queremos cargar dentro del include)

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center|top"
    android:orientation="vertical" >

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/guardar" />

    <include
        android:layout_width="wrap_content"
        android:layout_height="411dp"
        android:layout_marginLeft="136dp"
        android:layout_marginTop="186dp"
        layout="@layout/disenio_a_incluir" />

</LinearLayout>
```

¿Siempre es necesario añadir en include un merge?

No, pero es muy recomendable, ya que así quitamos un nivel de profundidad al árbol (tal como está el ejemplo, el include es lo mismo que tener el contenido del anterior merge, pero sin las etiquetas ni merge ni include, ya que se haría la sustitución del código del merge al include)

Atributos de las View

Se ha visto cómo crear los elementos. Todavía no hemos entrado en detalle de cómo se les da un poco de forma con los atributos.

¿Todas las Views tienen atributos?

Sí, cada View tiene sus propios atributos dependiendo de cuál sea. Por ejemplo, TextView tendrá el atributo "android:textSize" para agrandar el tamaño del texto; sin embargo, ImageView no tendrá ese atributo, tendrá otros

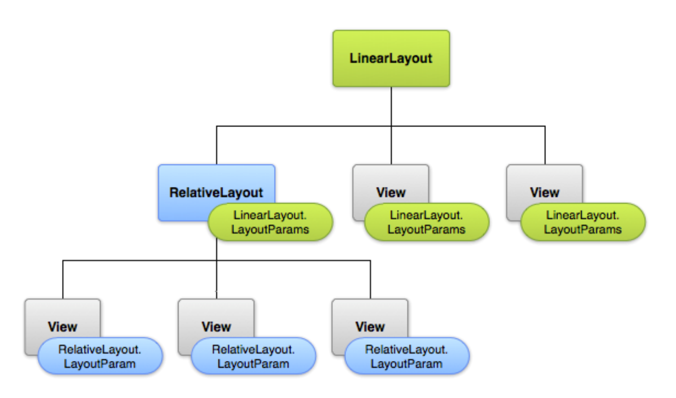
¿Son los mismos atributos dependiendo de cuál sea el padre de la View?

Existen los atributos de la View. Por otro lado, los atributos del padre -el ViewGroup contenedor- de la View que variarán dependiendo del padre.

Las características de los atributos heredados de un padre ViewGroup son:

- Nombrados como "**layout_algo**", como: layout_width, layout_centerHorizontal, layout_marginBottom, etc
- **LayoutParams**: atributos de una View que tiene que definir, que son implantados por su ViewGroup padre. Son propiedades que especifican el tamaño y la posición para cada View hija respecto a su ViewGroup padre
- Todos han de definir: **layout_width** (anchura) y **layout_height** (altura). Se pueden definir por tamaño exacto (ejemplo: 10dp), o una constante como **wrap_content** (se ajusta a su contenido), o fill_parent o **match_parent** (se dimensiona tan grande como su padre se lo permita)
- También todos tienen la opción de establecer margins (márgenes)

El ejemplo del árbol de la imagen de la derecha, nos viene a decir que las Views que están en el segundo nivel del árbol tendrá los atributos del "LinearLayout" que los contiene (fíjate que también los tiene el RelativeLayout). Y el tercer nivel del árbol, las Views tendrán atributos que heredan del padre, que serán propios del "RelativeLayout" (y también, date cuenta que no les llegan los atributos del LinearLayout que está dos niveles por encima; es decir, solo heredan los del padre inmediato)



¿Qué tiene en especial el atributo "android:id"?

Es un atributo global que aparece en todas las Views. Es un identificador único en el árbol. Cuando se compila la App este id se referencia como un entero en el fichero R.java para poder ser buscado desde Java.

Existen dos tipos de id que se pueden definir en XML:

- **android:id="@+id/button_id"** : la arroba "@" indica al conversor de XML que lo identifique como un id de un recurso. El más "+" significa que es un recurso nuevo que debe ser creado y añadido a los recursos (es decir, al fichero "R.java"). Se busca desde Java con: **R.id.boton_id**
- **android:id="@android:id/empty"** : sin el "+" y con android: delante, hace referencia a un id recurso de Android. Se busca desde Java con: **android.R.id.empty**

¿Si tengo que poner los mismos atributos a varias Views, tengo que ponerlos uno a uno en todas?

No es necesario, para eso están los estilos. Podremos asignar un estilo a una View o a un conjunto de Views, de este modo aplicar una serie de atributos que se repitan a las Views que elijamos (como que todos los textos de la aplicación sean de color azul).

¿Para qué sirven los atributos que empiezan con el espacio de nombres "tools" en las Views?

Únicamente sirven para ayudar a quien diseña el fichero de diseño XML. Lo veremos más adelante.

Referencias:

- http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html
- http://developer.android.com/guide/topics/ui/themes.html
- http://developer.android.com/guide/topics/resources/style-resource.html

Atributos para diseñadores en tiempo de diseño tools:xxx

¿Para qué sirven?

En los diseños XML, en espacio de nombres reservado "tools" únicamente sirve para guardar información que no afectará a las vistas de diseño del usuario, sino están destinadas a facilitar la vida a los diseñadores de los ficheros de diseño XML. Además, esta información de la etiqueta "tools" no será empaquetado a la hora de ejecutar la aplicación. Es decir, guarda información que no afecta visualmente a las Views en tiempo de ejecución (al ejecutar el programa en el emulador o en un dispositivo físico), pero sí en tiempo de diseño (mientras, como desarrolladores, diseñamos las vistas en Eclipse o en Android Studio).

¿Cuál es su espacio de nombres?

Para utilizarlo hay que declarar el espacio de nombres en la View raíz del fichero de diseño XML.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.jarroba.miapp.MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Una texto que debería Lint marcar como Hardcoded, pero no se
marca por usar tools:ignore con HardcodedText"
        tools:ignore="HardcodedText"
     />

</RelativeLayout>
```

Referencias:

- http://tools.android.com/tech-docs/tools-attributes
- http://developer.android.com/tools/debugging/improving-w-lint.html

- http://tools.android.com/tips/layout-designtime-attributes

¿Cuáles son?

tools:ignore

Para que Lint ignore alguna comprobación. Por ejemplo el valor HardcodedText, que se usa para evitar que Lint nos recuerde con un Warning (en Eclipse subrayado de amarillo) los textos que introduzcamos directamente y no en el fichero "strings.xml"

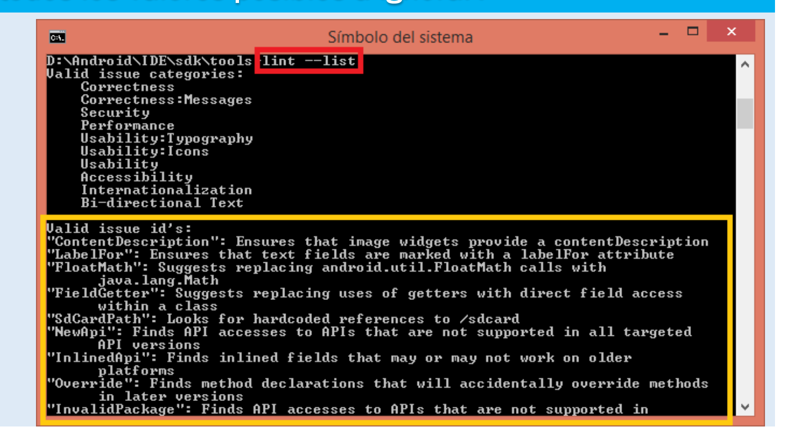
```
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Una texto que debería Lint marcar como Hardcoded, pero no se marca
por usar tools:ignore con HardcodedText"
      tools:ignore="HardcodedText"
      />
```

¿Dónde puedo encontrar todos los valores posibles a ignorar?

Estos se pueden ver en la consola de comandos. Para ello:

1. Abrimos la **consola** de comandos
2. Llegar hasta la **carpeta tools del SDK** de Android y escribir:

```
lint --list
```



tools:targetApi

Para indicar a partir de cual nivel del API se utilizará la View. Por ejemplo GridLayout, no funcionaría en versiones anteriores a la 14. Simplemente es un recordatorio para saber que versión de Android es la mínima que soporta.

```
<GridLayout
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     tools:targetApi="ICE_CREAM_SANDWICH" >
</GridLayout>
```

tools:context

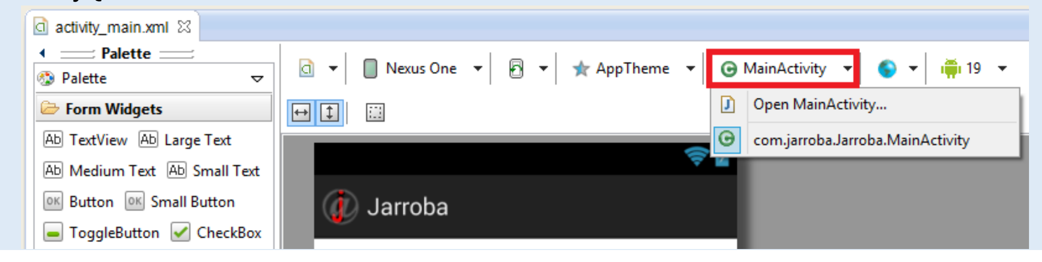
Se pone en la View raíz del fichero de diseño XML, para recordar que Activity la asociará. De este modo el editor de diseños podrá poner el tema de la Activity (el cual estará indicado en el AndroidManifest.xml, donde se asocia con la Activity, no con el fichero de diseño XML) mientras diseñamos nuestra vista (este tema es solo en tiempo de diseño, no el que verá el usuario final). Se puede utilizar el punto como prefijo, al igual que en el AndroidManifest.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

</RelativeLayout>
```

En el editor gráfico ¿Existe una manera fácil de asignar esta propiedad?

Sí, existe la opción de elegir o de cambiar la propiedad en el icono de C en círculo verde (de clase Java). Una ventaja de asignarlo, es que podremos abrir directamente desde aquí la Activity que la asociará pulsando sobre "Open ActivityQueLaAsociara…".



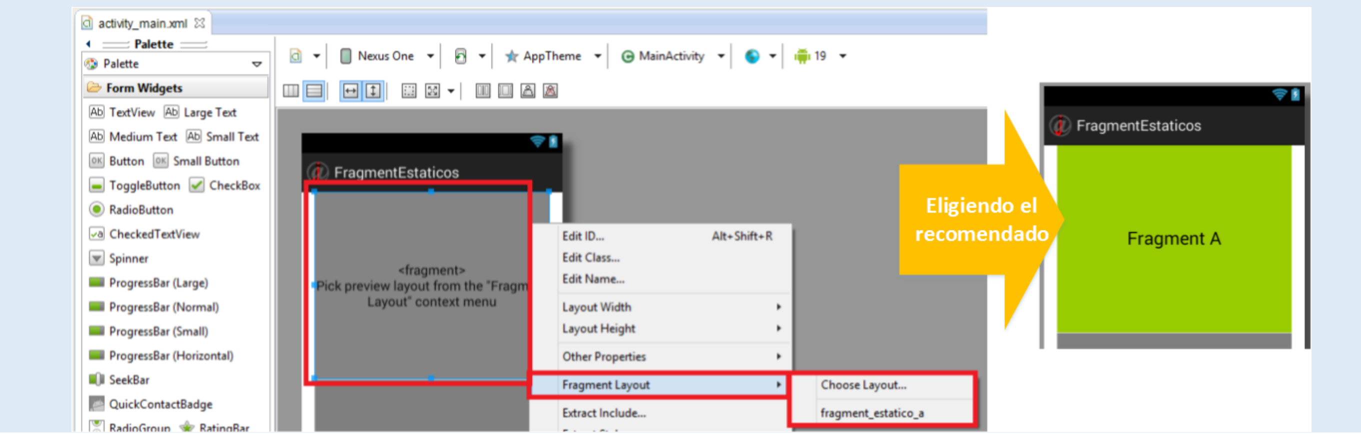
tools:layout

Muy útil para diseñar Fragments. Se pone en la etiqueta <fragment> y se utiliza para recordar que fichero de diseño XML se tendrá que dibujar, pero solo para ver cómo queda para el diseñador, no para el usuario. Podremos poner cualquiera, no teniendo que corresponder con el del Fragment propiamente, aunque será lo más normal. Cuando ejecutemos la aplicación, el diseño se determinará por el que hayamos puesto en el onCreateView()

```
<fragment
    android:id="@+id/fragment_A"
    android:name="com.jarroba.fragmentestaticos.EstaticoFragmentA"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="2"
    tools:layout="@layout/disenio_fragment" />
```

En el editor gráfico ¿Existe una manera fácil de asignar esta propiedad?

Sí. Seleccionando el Fragment en la vista previa, luego pulsando el botón derecho se nos desplegará el panel de opciones. Ahí pulsaremos "Frame Layout" y luego elegiremos el que nos recomienda (Basado en el que hayamos asociado en el Fragment en el onCreateView(), por lo que será probable que nos interese) o cualquier otro diseño.



tools:locale

Para indicar que un fichero de "strings.xml" corresponde con un idioma y opcionalmente con una región. Útil para para indicar que idioma se utilizará en el fichero.

res/values-en/strings.xml

```
<resources xmlns:tools="http://schemas.android.com/tools" tools:locale="en">

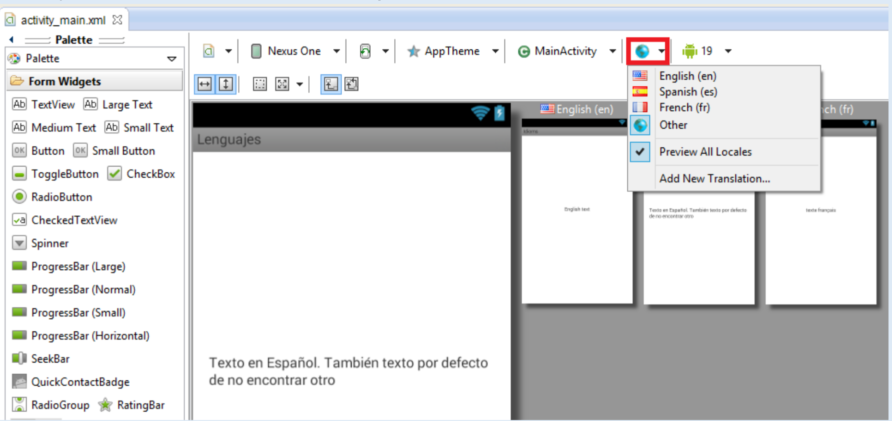
    <string name="app_name">Idioms</string>
```

```
    <string name="texto_del_textView">English text</string>

</resources>
```

En el editor gráfico aparecen los idiomas ¿Puedo gestionarlos desde aquí?

No. En el icono del planeta puedes cambiar el idioma para ver cómo se vería en otro idioma, siempre que tengas correctamente los ficheros "strings.xml" en la carpeta "res" y en cada carpeta "values-xx" (donde xx puede ser "es", "en", etc). O incluso, ver una vista previa de cómo se verían todas las vistas a la vez.



tools:listitem, tools:listheader, tools:listfooter

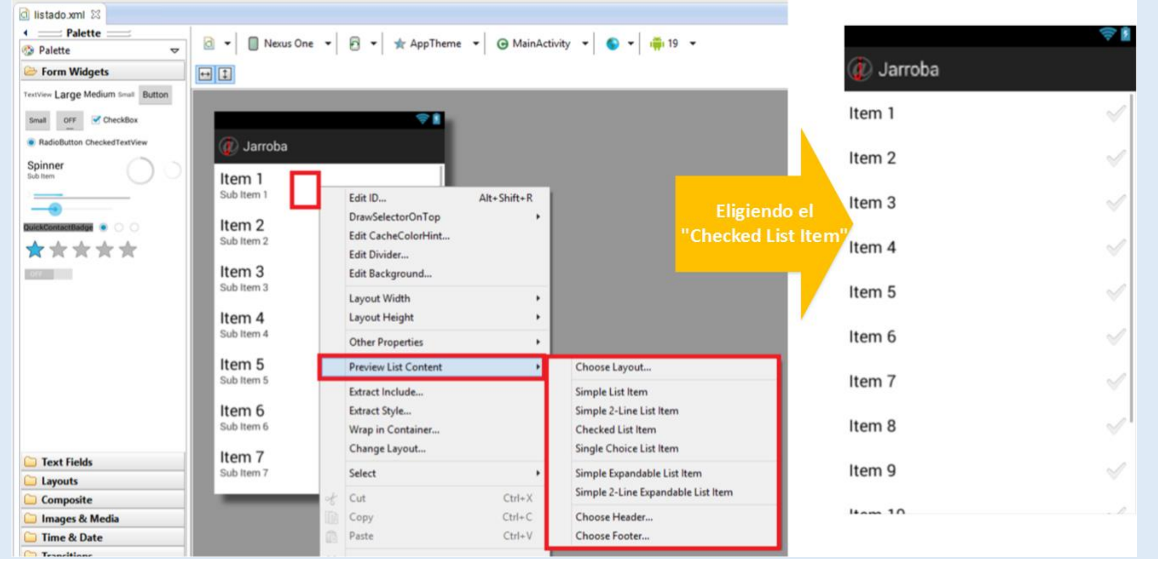
Para llenar de contenido falso al <ListView> y poder ver cómo queda

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.jarroba.Jarroba.MainActivity"
    tools:ignore="MergeRootFrame"
    tools:listitem="@android:layout/simple_list_item_checked" >

</ListView>
```

En el editor gráfico ¿Existe una manera fácil de asignar esta propiedad?

Sí. En donde tengamos un listado, sobre la vista previa podemos pulsar con el botón derecho del ratón, elegir "Preview List Content". Ahí elegir el que queramos que tome, o uno hecho por nosotros o uno ya hecho por los chicos de Google. También podremos elegir un encabezado o pie de lista.



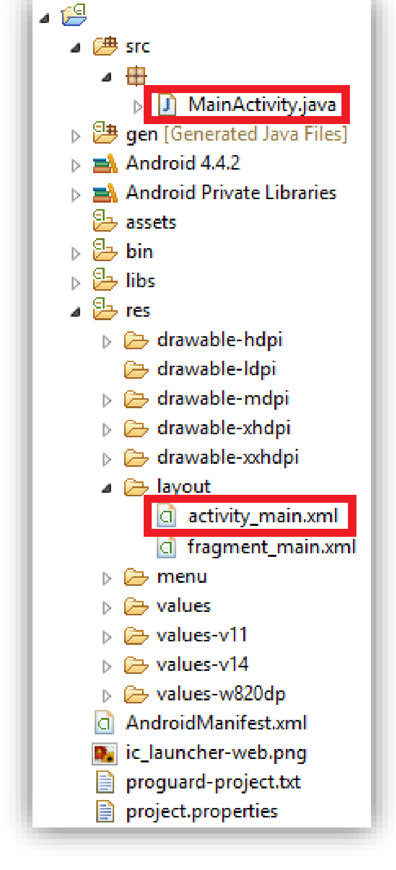
tools:showIn

Se pone en el View raíz del fichero para recordar en que <include> va a ir dibujado. Normalmente en se pone en un <merge> que suele utilizarse para incluirse en una etiqueta <include>. Permite apuntar al fichero de diseño XML que apuntará a este otro diseño, y en tiempo de diseño éste se incluirá en el exterior del diseño que lo rodea

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:showIn="@layout/disenio_que_incluye_con_include"
    >
</merge>
```

Asociar un diseño (Layout) a un Activity o un Fragment

¿Cómo se asocia un diseño a una Activity en Java?



Siempre se hace en el método onCreate() del ciclo de vida de la Activity (Recuerdo que una clase para ser Acitivity ha de heredar de Activity). Se asocia el fichero del diseño XML con el método setContentView() justo después de la llamada al padre (justo después del "super").

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
}
```

Este método setContentView() solicita un valor entero. Este valor ha de ser un identificador de recurso que apunte a la carpeta "layout" y dentro de esta al fichero que queramos asociar, como puede ser "activity_main.xml". Para ello utilizaremos un fichero especial llamado "R". Por lo que podremos llamar a nuestro diseño con "R.layout.activity_main" (Daremos esto de manera más profunda en el tema de Recursos).

¿Se asocia igual en un Fragmen en Java?

En Fragment siempre se hace en el método onCreateView() del ciclo de vida del Fragment (Recuerdo que éstas clases heredan de Fragment). Se asocia el fichero del diseño XML con el método inflate() del objeto LayoutInflater.

Ojo a las diferencias respecto a cómo se asocia un diseño en una Activity y como se asocia en un Fragment.

El inflate() no debe adjuntar el diseño al diseño raíz.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle savedInstanceState) {
      View rootView = inflater.inflate(R.layout.fragment_main, container, false);
      return rootView;
}
```

¿Cómo funciona inflate()?

Antes de explicar este método, indicar que se utiliza para mucho más que en solo Fragments (también se usa con Adapter, Menu, entre otros muchos). Lo explicamos aquí por ser la primera aparición y dejarlo claro.

También decir, que esta explicación es la visión del autor de este libro (después de años trabajando con ello, de haber investigado y deducido una explicación razonable). Ya que en ningún lado parece existir una explicación clara o un consenso de qué es lo que hace realmente el método inflate() -casi ni los propios de Google ya que no lo explican de una manera concisa, lo dan por dado. Sin embargo, aquí lo voy a explicar con detalle.

Una cosa que hay que tener muy clara del método inflate(), que pertenece al objeto de la clase LayoutInflater, es –lo que lleva demasiado a confusión- que inflar no se refiere a insertar unas Views en un ViewGroup, sino a poner los atributos de diseño del que podría ser el ViewGroup padre. Aquí seguramente surja la duda ¿Pero si cuando inflo se añaden las Views al ViewGroup? Y es que inflate() aparte de inflar, también tiene la opción de adjuntar (adjuntar es un valor añadido a la función inflate(), pero para nada adjuntar se refiere a inflar).

Veámoslo en un ejemplo. Supongamos que tenemos un diseño que dibuja un texto como el siguiente:


Ejemplo de código res/layout/adjuntar_en_un_viewgroup.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/otro_fichero_de_dise_o_xml"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>
```


Y lo que queremos es adjuntar el diseño anterior al RelativeLayout de este otro diseño:


Ejemplo de código res/layout/fichero_de_disenio.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/otra_view" />

    <RelativeLayout
        android:id="@+id/RelativeLayout_contenedor"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:orientation="vertical" >
    </RelativeLayout>

</LinearLayout>
```

Como queremos es que la estructura del primer diseño se adjunte al ViewGroup del segundo diseño (algo parecido a lo que se hacía con las Views Merge e Include, pero en dinámico, desde Java):

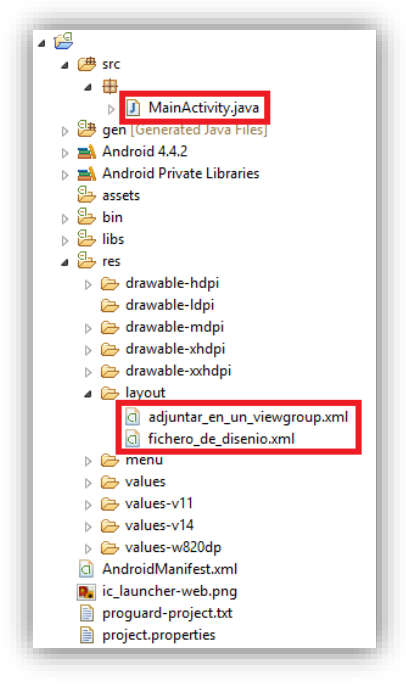
```
public class MainActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.fichero_de_disenio);

                RelativeLayout contenedor = (RelativeLayout) findViewById(R.id.RelativeLayout_contenedor);

                LayoutInflater inflater = LayoutInflater.from(this);

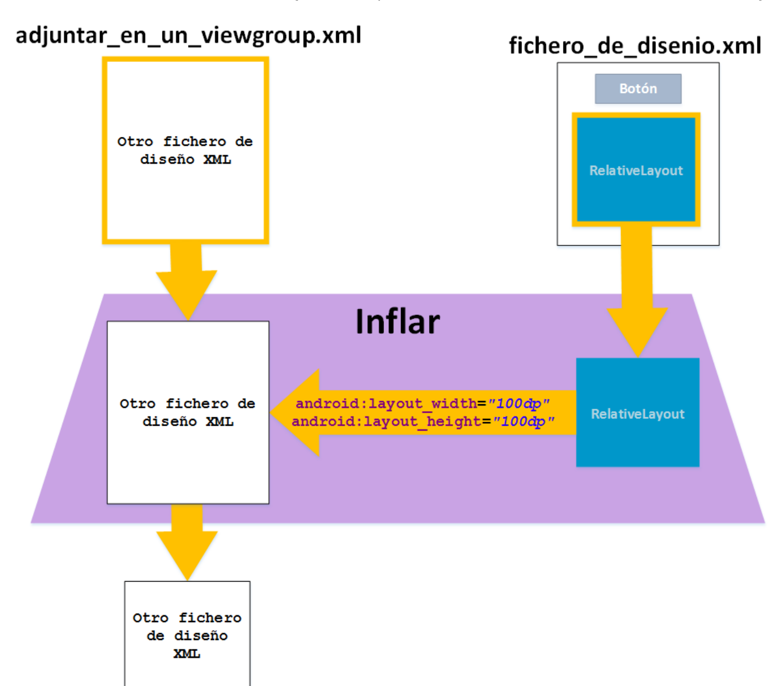
                View laViewInflada = inflater.inflate(R.layout.adjuntar_en_un_viewgroup, contenedor, true);
        }
}
```

Este código anterior nos da la siguiente estructura de carpetas de ejemplo (el código de este ejemplo es un proyecto completo funcional, puedes crear un nuevo proyecto probarlo):



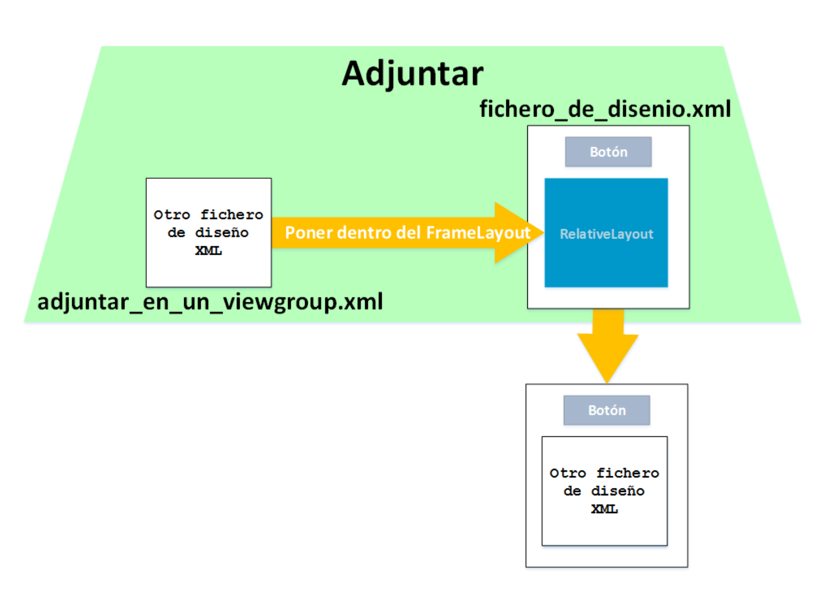
Llegó el momento de tener muy claras las diferencias entre inflar y adjuntar:

- **Inflar:** Por contra a la intuición, NO infla al ViewGroup contenedor con las Views que le pasamos (el fichero de diseño XML que queremos adjuntar al contenedor). Lo que se infla es la nueva jerarquía de Views (el diseño) que les pasamos con los atributos de diseño del padre (con los atributos del ViewGroup contenedor). Es decir, le decimos a las nuevas Views el tamaño que tienen que tener para que quepan en el padre, la gravedad, entre otros (dependerán del ViewGroup que usemos para inflar: LinearLayout, FrameLayout, Fragment, etc). Es por lo que se deduce que no dibuje en pantalla lo inflado y por tanto no pueda recibir eventos.



En el ejemplo, le pasaríamos al método inflate() el fichero de diseño de “adjuntar_en_un_viewgroup.xml” y el RelativeLayout del que será el contenedor (fíjate en los parámetros primero y segundo que se le pasa al método inflate()). De este modo, se infla la estructura de views del fichero con los parámetros del contenedor que se le pasan del que será el padre y contenedor.

- **Adjuntar:** añade el nuevo diseño que se infló a la ViewGroup y ahora sí lo muestra en pantalla y podrá recibir eventos. El tercer parámetro del método inflate() indica si queremos que lo adjunte inmediatamente o no (por ejemplo, en Fragment no interesa, pues lo hará el propio Fragment en otro momento, por eso lo retornamos el objeto LayoutInflater en el método onCreteView()).



Volviendo al ejemplo. El tercer parámetro del método inflate() indica si queremos que la estructura que antes hinchamos se adjunte al contenedor (true) o no (false).

Por lo que después de adjuntar (que no de inflar), a nivel interno nos habrá generado la siguiente estructura que el usuario podrá ver y responderá a eventos:



Ejemplo de código

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/otra_view" />

    <RelativeLayout
        android:id="@+id/RelativeLayout1"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:orientation="vertical" >

        <LinearLayout
            android:id="@+id/LinearLayout1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/textView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/otro_fichero_de_dise_o_xml"
                android:textAppearance="?android:attr/textAppearanceLarge" />

        </LinearLayout>

    </RelativeLayout>

</LinearLayout>
```


Ya los hemos visto por encima. Ahora definimos los tres parámetros que solicita el método inflate() (hay otras sobrecargas de este método, pero explicamos el de tres parámetros que se suele utilizar más):

1. **Recurso de diseño (resource):** La referencia de R del diseño que queramos inflar.
2. **Contenedor raíz (root de tipo ViewGroup):** El contenedor cuyos parámetros de su diseño (todos sus "android:layout_xxxxx") inflarán al anterior recurso de diseño indicado en el parámetro anterior. Y si el

tercer parámetro lo permite con un true, se adjuntará el diseño anterior a este contenedor. Puede interesar pasar aquí un “null” si no queremos que se infle con ningún parámetro de ningún padre.

3. **Si adjuntar al contenedor que será la raíz (boolean):** Si es “true” indica si el diseño (primer parámetro) que es inflado por el ViewGroup contenedor (segundo parámetro) debe, además, adjuntar su jerarquía de Views a este contenedor. A veces interesa poner este parámetro a “false” porque nos interese solo inflar, debido a que se adjuntará lo inflado en otro momento diferente posterior (por eso devuelve una View inflada el método inflate()); el motivo suele ser que no se cree dos veces el mismo conjunto de Views (Como en el método onCreateView() de Fragment, donde se encarga el Fragment de adjuntarlo cuando deba; si lo ponemos a “true” nos aparecerá dos veces lo mismo)

Puede que me haya explayado un poco. Es necesario entender este concepto para no copiar y pegar código sin saber qué hace. Además, este método seguro que lo utilizarás en más de una ocasión.

¿El diseño hecho en un fichero XML para una Activity vale para un Fragment?

Sí, son intercambiables, se diseñan de igual manera (puede que interese cambiar la colocación de Views, o el diseño de una Activity partirlo en varios Fragments). Por el nuevo paradigma de uso de Fragments, recomiendo que el diseño (Botones, textos, etc) recaiga sobre los Fragments; y que el fichero de diseño asociado a los Activities tan sólo sirva para posicionar a los Fragments en pantalla.

Referencias:

- http://jarroba.com/infate-en-android-inflar-y-adjuntar-views/
- http://developer.android.com/reference/android/view/LayoutInflater.html#from(android.content.Context)
- http://developer.android.com/guide/components/fragments.html

Ejemplo de diseño de Activity con Fragments

Lo que haremos

Vamos a crear una aplicación en la que tendremos una Activity que contendrá dos Fragments estáticos simples -en un futuro les pondremos funcionalidad- cada uno tendrá un texto y un color.

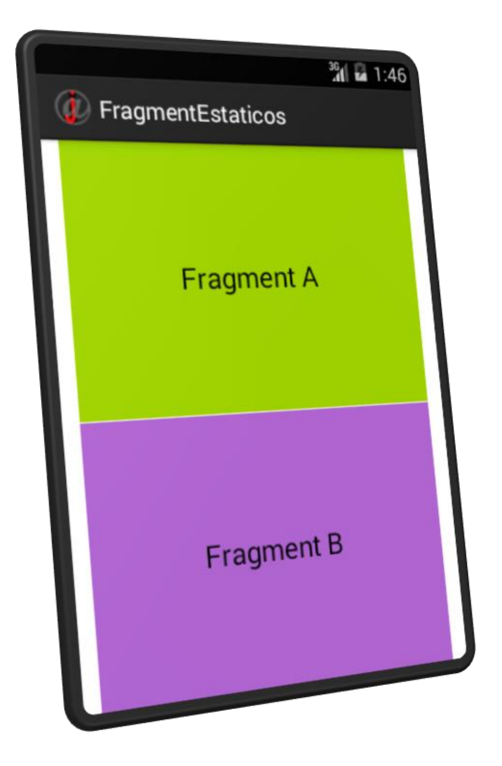


Ilustración 7 - Resultado final de tener una Activity con dos Fragments

¿No hay ejemplo de diseños con Activity sin Fragments?

Si necesitas repasar como asociar un diseño a una Activity de manera simple (en este ejemplo lo haremos un poco más a lo grande) echa un vistazo a los ejemplos hechos anteriormente. En esos ejemplos anteriores, podrás ver como se trata la asociación simple de un diseño con una Activity. De cualquier manera, en este ejemplo haremos lo mismo y más, pues trabajaremos con Fragments.

Si estás empezando en Android no te recomiendo que dediques muchos esfuerzos a trabajar solo con Acitivities y sus diseños, ya que ahora todo se basa en la arquitectura de Fragments por lo que ya vio. Cuanto antes comiences a trabajar con Fragments mejor que mejor. Si este leyendo este libro es porque quieres aprender a trabajar con Fragments intensamente ☺

Es muy sencillo, pero tenemos que entender la arquitectura. Para realizar esto necesitamos los diseños siguientes:

- Diseño para la Activity: la cual contendrá dos Fragments
- Diseño para el Fragment A: un fondo verde
- Diseño para el Fragment B: un fondo morado

Hemos visto que necesitamos una Activity y dos Framents. Pues serán las que tengamos en código Java también. La Activity asociará su diseño, y cada Fragment asociará su respectivo diseño.

Ahora bien, en el diseño de la Activity, como pondremos dos Fragmetns estáticos, los pondremos directamente con las etiquetas <fragment>. De este modo, y como los Fragments nunca variarán, podremos indicar en los atributos del mismo diseño a que código Java que herede de Fragment irá asociado cada uno (que será el encargado de darle funcionalidad). Esto lo haremos mediante el atributo "android:name".

Por tanto y en total, lo que haremos será lo que se muestra en la siguiente figura:

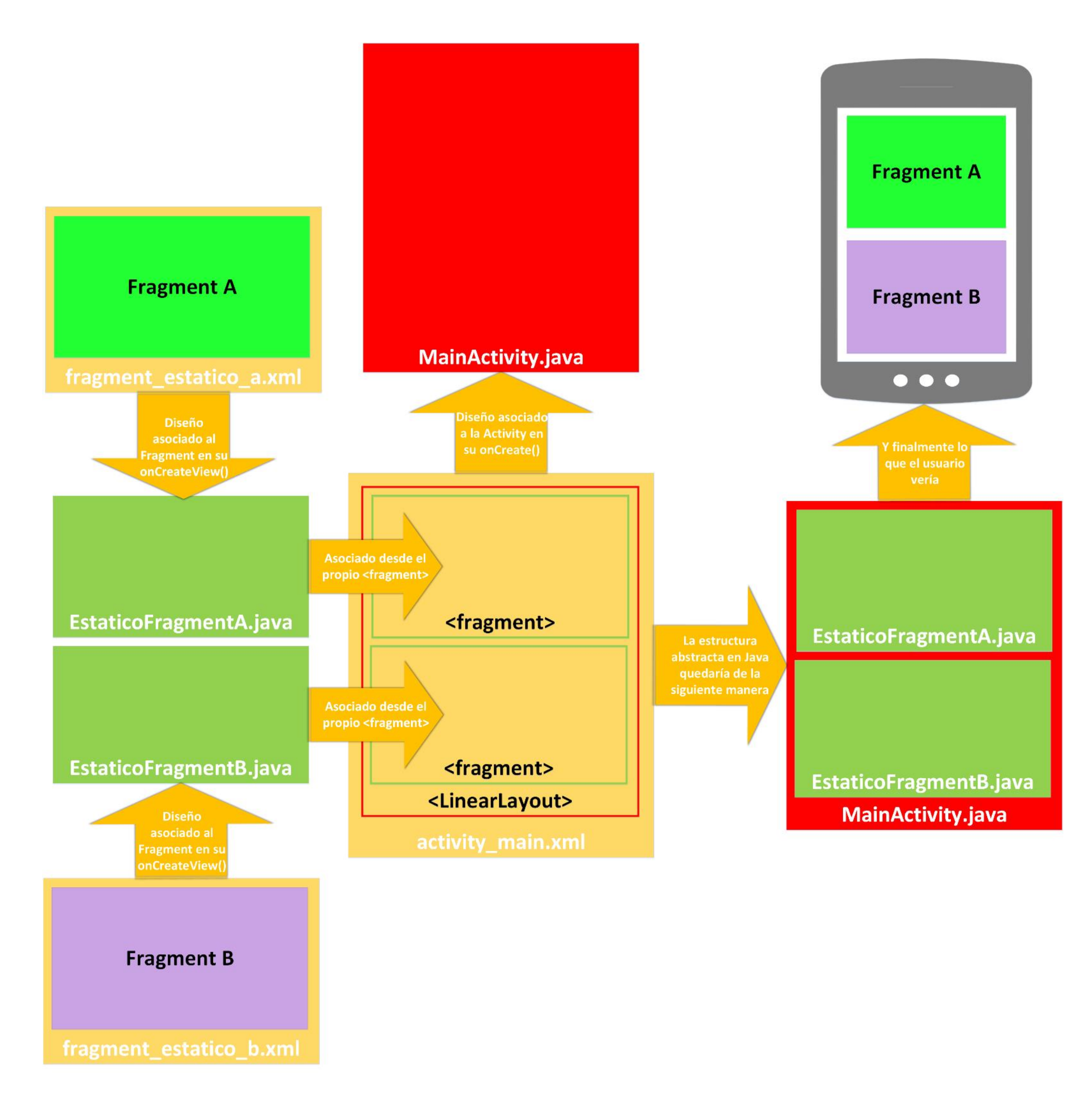
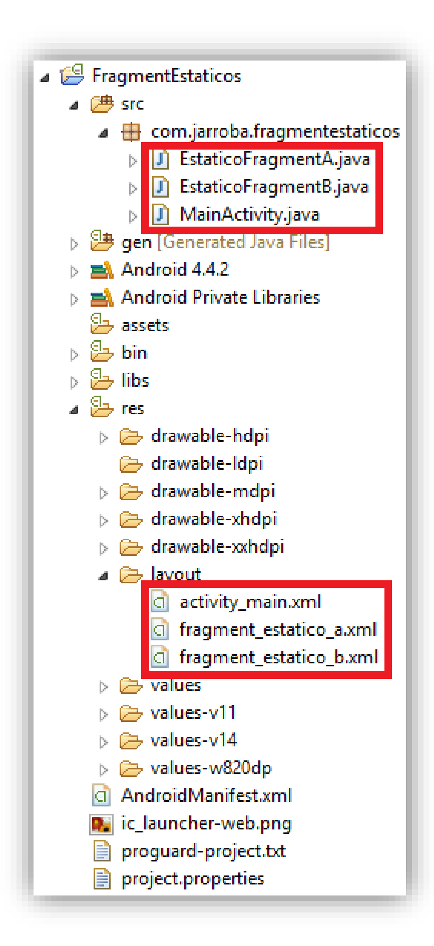


Ilustración 8 - Trabajar con Fragments estáticos

Para realizar crear este proyecto recomiendo elegir un mínimo de SDK de: "API 16: Android 4.1 (Jelly Bean)".

Ya tenemos todo para empezar a trabajar.



EstaticoFragmentA.java

```
public class EstaticoFragmentA extends Fragment {

	public EstaticoFragmentA() {
	}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_estatico_a, container, false);
		return rootView;
	}
}
```

EstaticoFragmentB.java

```
public class EstaticoFragmentB extends Fragment {

	public EstaticoFragmentB() {
	}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_estatico_b, container, false);
		return rootView;
	}
}
```

MainActivity.java

```
public class MainActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
	}

}
```

res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:baselineAligned="false"
    android:divider="?android:attr/dividerHorizontal"
    android:orientation="vertical"
    android:showDividers="middle" >

    <fragment
        android:id="@+id/fragment_A"
        android:name="com.jarroba.fragmentestaticos.EstaticoFragmentA"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="2"
        tools:layout="@layout/fragment_estatico_a" />

    <fragment
        android:id="@+id/fragment_B"
        android:name="com.jarroba.fragmentestaticos.EstaticoFragmentB"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="2"
        tools:layout="@layout/fragment_estatico_b" />

</LinearLayout>
```

res/layout/fragment_estatico_a.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Fragment A"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>
```

res/layout/fragment_estatico_b.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_purple" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Fragment B"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>
```

Diseños para distintos tipos de pantallas, y por tanto de dispositivos



¿Tengo que compilar una aplicación para cada tipo de pantalla que existe?

No, ¡Por el bien de tu yo futuro ni se te ocurra!, tendrías el código repetido en un montón de sitios y un cambio implicaría cambiarlos todos los proyectos.



Android nos ayuda con la reutilización del código permitiéndonos, con un mismo proyecto, crear distintos diseños para todos los tipos de pantallas que necesitemos.

Para entender las siguientes preguntas, te recomiendo que repases el tema de Recursos si algo no lo ves claro.

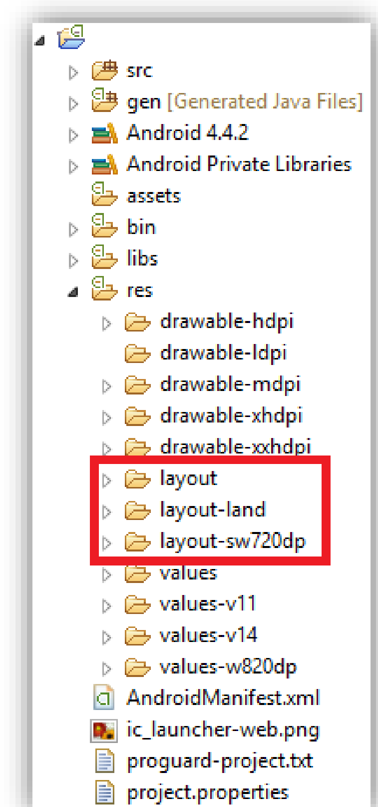
¿Cuántos tipos de pantalla existen en Android?

Hoy son finitos, cerca de mil (por decir una cifra aleatoria muy alta sin más base que la intuición). La idea es que tiende a infinito. Puede que esta respuesta te parezca extraña, saldrás de dudas en la siguiente pregunta.

¿A qué nos referimos a tipos o múltiples pantallas?

Cuando nos dicen múltiples pantallas lo más seguro es que se te venga a la cabeza las pulgadas, es decir, el tamaño de la diagonal. Y es cierto, es un factor que establece el tipo de pantalla, pero no el único, dependerá de su configuración. Existen cuatro factores determinantes:

- Tamaño: las pulgadas que tiene la pantalla.
- Densidad: cantidad de píxeles concentrados en un área.
- Orientación: si está la pantalla en posición vertical (portrait) u horizontal (landscape).
- Relación de aspecto (Aspect ratio): proporción entre el alto y la anchura de la pantalla; es decir, si el rectángulo que describe la pantalla está más o menos estirado.



¿Cómo sabe Android que diseño aplicar a cada tamaño de pantalla?

Por la carpeta "layout" donde se encuentre el recurso. Siempre dentro de la carpeta "res" podremos crear varias carpetas con prefijo "layout" seguido de un guion "-" y un prefijo como "land"; quedando "layout-land".

Cuando la aplicación se ejecute en un dispositivo, Android informará a la aplicación del tipo de pantalla. Entonces buscará los recursos de diseño apropiados en la carpeta "layout", cuyo prefijo cumpla las condiciones. De no encontrar una que cumpla las condiciones, o si existe la carpeta no tuviera el recurso requerido, siempre buscará en la carpeta que no tiene prefijo y se llama "layout. Por lo que siempre ha de existir la carpeta que se llama únicamente "layout" con todos los recursos posibles; es decir, es la carpeta que se podría llamar "para el resto de los tipos de pantallas" o carpeta por defecto.

¿Cuáles son las configuraciones típicas y las recomendables?

Debido a que Android puede ser ejecutado en distintos dispositivos, de diferentes tamaños, formas de pantalla y densidades, hemos recogido una tabla con los más típicos.

Indicar que los datos del siguiente cuadro son muy comunes, pero aproximados y pueden variar; recogidos por la experiencia e investigación del autor de este libro, para mantener una guía a la que poder acudir en caso de duda.

Dispositivo	Pulgadas	Forma	Densidades típicas	Carpeta "layout" recomendada
Reloj	1 a 3	Cuadrado o círculo	(Esperando al lanzamiento de Android Wareables)	Por determinar, posiblemente sea tamaño inalterable
Smartphone	3 a 5	Rectángulo alargado	240x320 ldpi, 320x480 mdpi, 480x800 hdpi	layout-sw320dp
Phablet	5 a 7	Rectángulo alargado	480x800 mdpi	layout-sw480dp
Tablet	7 a 12	Proporción aurea	600x1024 mdpi, 720x1280 mdpi, 800x1280 mdpi	Para Tablets 7": layout-sw600dp Para Tablets 10": layout-sw720dp Para generalizar: **layout-w820dp**
Pantalla de ordenador	12 a 30	Rectángulo alargado o proporción aurea	1920x1080	Por determinar
Televisión	30 a 80	Rectángulo alargado	(Depende de la televisión y el reproductor multimedia)	Por determinar
Proyector	80 a 300	Rectángulo alargado	(Depende del proyector y el reproductor multimedia)	Por determinar

Recomiendo mínimo un diseño base en la carpeta "layout" -que servirá para todos- y uno para Tablet (el que recomienda Android hoy día es "layout-w820dp"). Con estos tendremos cubiertos muchos de los diseños. No quita que podamos hacer otros; es más, estará mejor tener los apropiados para la comodidad de uso de la aplicación en los diferentes tipos de pantalla.

¿Existe otra manera de cambiar las pantallas?

Si no quieres tener varias carpetas "layout" con sufijos, puedes hacer todos los diseños en la carpeta que se llama "layout" con diferente nombre. Y en la carpeta "values" correspondiente crear un alias. Ya no lo recomiendo así por temas de estructuración del proyecto, pero si te interesa, tienes un ejemplo en: http://jarroba.com/programar-fragments-fragmentos-en-android/

Referencias:

- http://developer.android.com/guide/practices/screens_support.html

Ejemplo de diseño para varias pantallas

Lo que haremos

Depende de la pantalla que tengamos queremos un diseño u otro. Es decir, podremos querer un diseño diferente para Tablets, otro para Smartphones, otro para televisiones, otro para pantallas en modo paisaje (lanscape), etc.

En este ejemplo para simplificar vamos a hacer un diseño para las pantallas cuando están en modo portrait (sujetamos el dispositivo en posición vertical) y otra para cuando la pantalla está en landscape (colocando al dispositivo en posición horizontal). Cabe decir, que aunque este ejemplo el diseño varíe dependiendo de cómo el usuario sujete la pantalla, para hacer lo mismo pero para diferentes tamaños de pantalla como diferenciar entre Tablet o Smartphone, se hace exactamente igual, variando los sufijos de la carpeta “layout”.

Reutilizaremos casi todo el código del ejemplo anterior, aunque podríamos aplicarlo a cualquier diseño que queramos y no solo a este que contiene Fragments.

Queremos que en modo portrait se vea un único Fragment (en la imaginación supondremos que este Fragment tiene tantas cosas, que para el usuario le resultaría cómodo manejar un solo Fragment en vez de dos; de algún modo llegaríamos al segundo, por ejemplo pulsando un botón)



Ilustración 9 - Diseño para posición Portrait

Pero si el usuario colocara el dispositivo en posición landscape, el tamaño sería suficiente como para mostrar los dos Fragments (en este caso y en esta posición imaginamos que el contenido de cada Fragment tiene lo suficiente como para ser cómodo para el usuario).

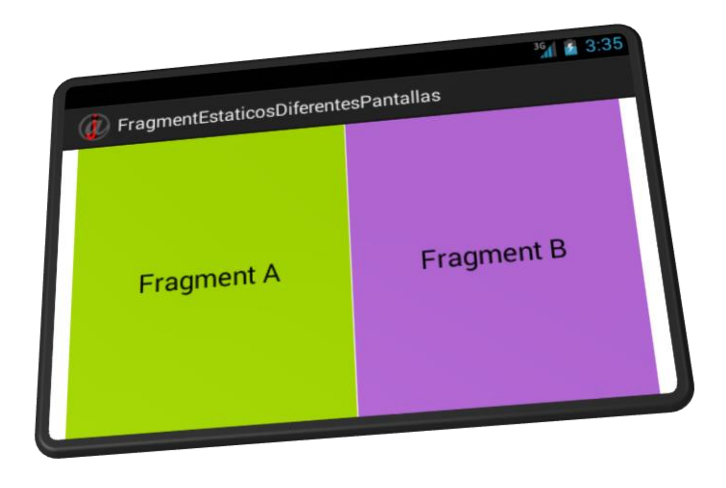
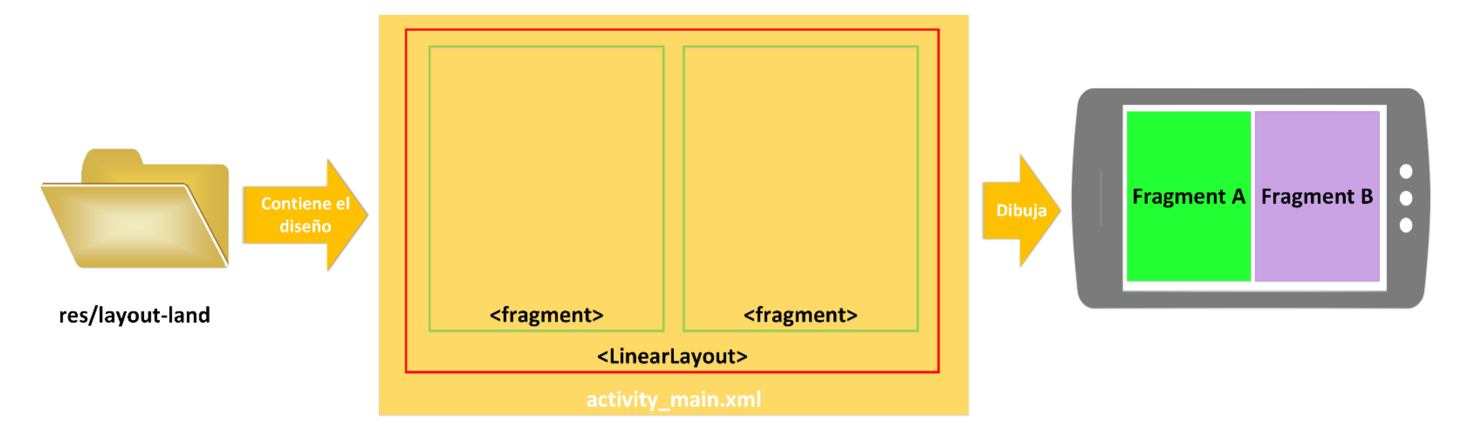


Ilustración 10 - Diseño para posición Landscape

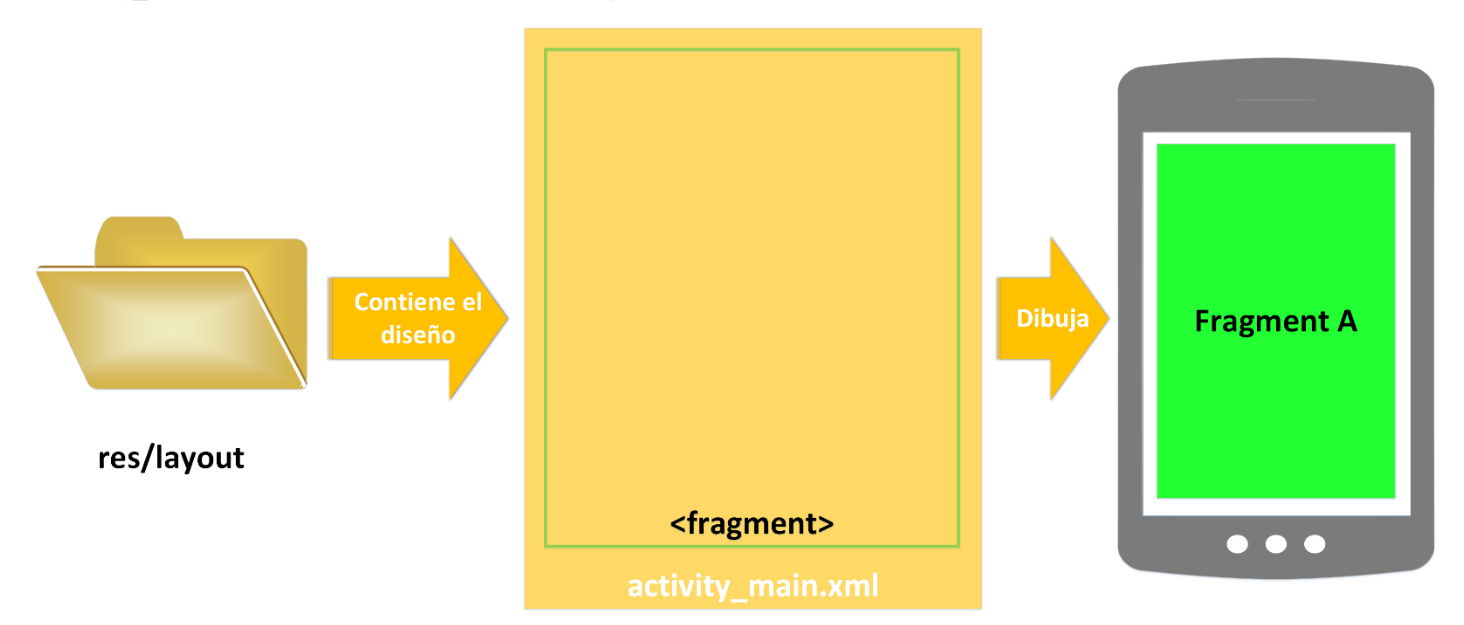
Para realizar esto utilizaremos las carpetas “layout” para realizar los múltiples diseños.

Queremos que exista un diseño para la posición “landscape”. Por lo que crearemos una nueva carpeta dentro de “res”, cuyo nombre será “layout-land” (la creamos o creamos el fichero XML con el elemento que cualifica llamado “Orientation” y eligiendo la propiedad “Landscape”). De esta manera le decimos a Android que cuando esté en posición “landscape” tome los diseños de esta carpeta. Y si está la pantalla de cualquier otro modo, tomará los diseños de la carpeta “layout” (al no tener prefijo la carpeta “layout”, estamos indicando que si Android no encuentra el diseño en las otras carpetas, busque en esta en último lugar).

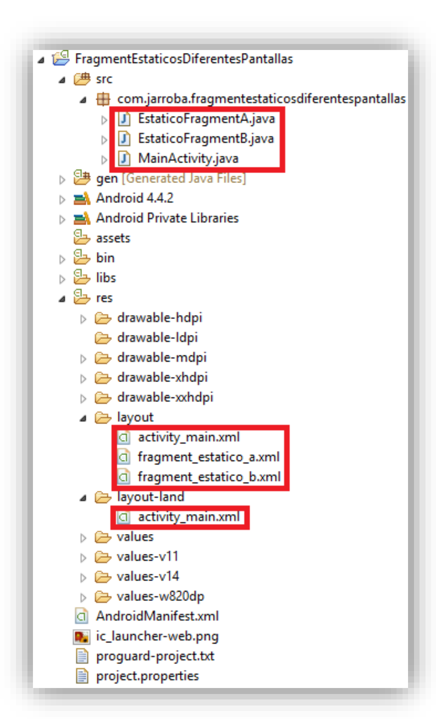
Así crearemos la carpeta “layout-land”, a la que crearemos el fichero llamado “activity_main.xml”.



Dentro de la carpeta “layout” crearemos un fichero con el mismo nombre que el anterior, es decir, llamado “activity_main.xml”. Éste fichero tendrá un código diferente al anterior.



De este modo tendremos dos diseños diferenciados para cada tipo de pantalla. Será Android el que se encargue de seleccionar el apropiado, dependiendo de la posición pantalla del dispositivo.



Reutilizamos código: El código que no está aquí se encuentra en el ejemplo anterior.

res/layout/activity_main.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_A"
    android:name="com.jarroba.fragmentestaticosdiferentespantallas.EstaticoFragmentA"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout="@layout/fragment_estatico_a" />
```

res/layout-land/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:baselineAligned="false"
    android:divider="?android:attr/dividerHorizontal"
    android:orientation="horizontal"
    android:showDividers="middle" >

    <fragment
        android:id="@+id/fragment_A"
        android:name="com.jarroba.fragmentestaticosdiferentespantallas.EstaticoFragmentA"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="2"
        tools:layout="@layout/fragment_estatico_a" />

    <fragment
        android:id="@+id/fragment_B"
        android:name="com.jarroba.fragmentestaticosdiferentespantallas.EstaticoFragmentB"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="2"
        tools:layout="@layout/fragment_estatico_b" />

</LinearLayout>
```

Utilizar las Views desde Java

¿Cómo encuentro desde Java una View para poder utilizarla en una Activity?

Lo primero que hay que hacer es encontrar nuestra View por su atributo “id” con el método **findViewById().** Esta id la encontraremos mediante el fichero “R.java”.

Un ejemplo para encontrar un TextView puede ser este:

```
TextView miTexto = (TextView) findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);
```

¿Cómo se utiliza el fichero “R.java”?

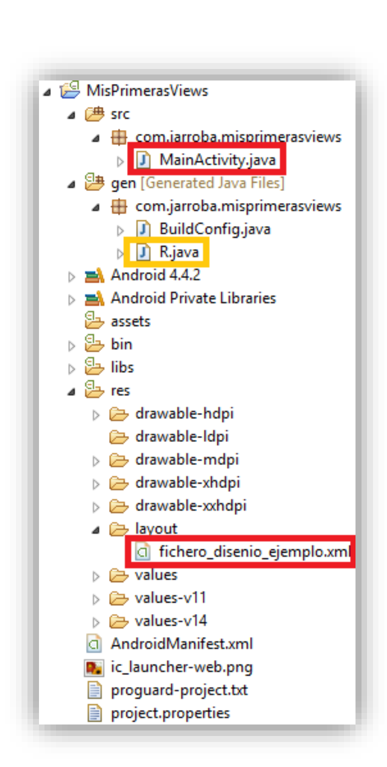
Veámoslo con un ejemplo. Para ello vamos a crear un diseño nuevo con un TextView. Supongamos que tenemos un diseño que dibuja un texto como el siguiente:

Ejemplo de código de diseño de un Layout XML llamado res/layout/fichero_disenio_ejemplo.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout_contenedorDeMisViews"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="${packageName}.${activityClass}" >

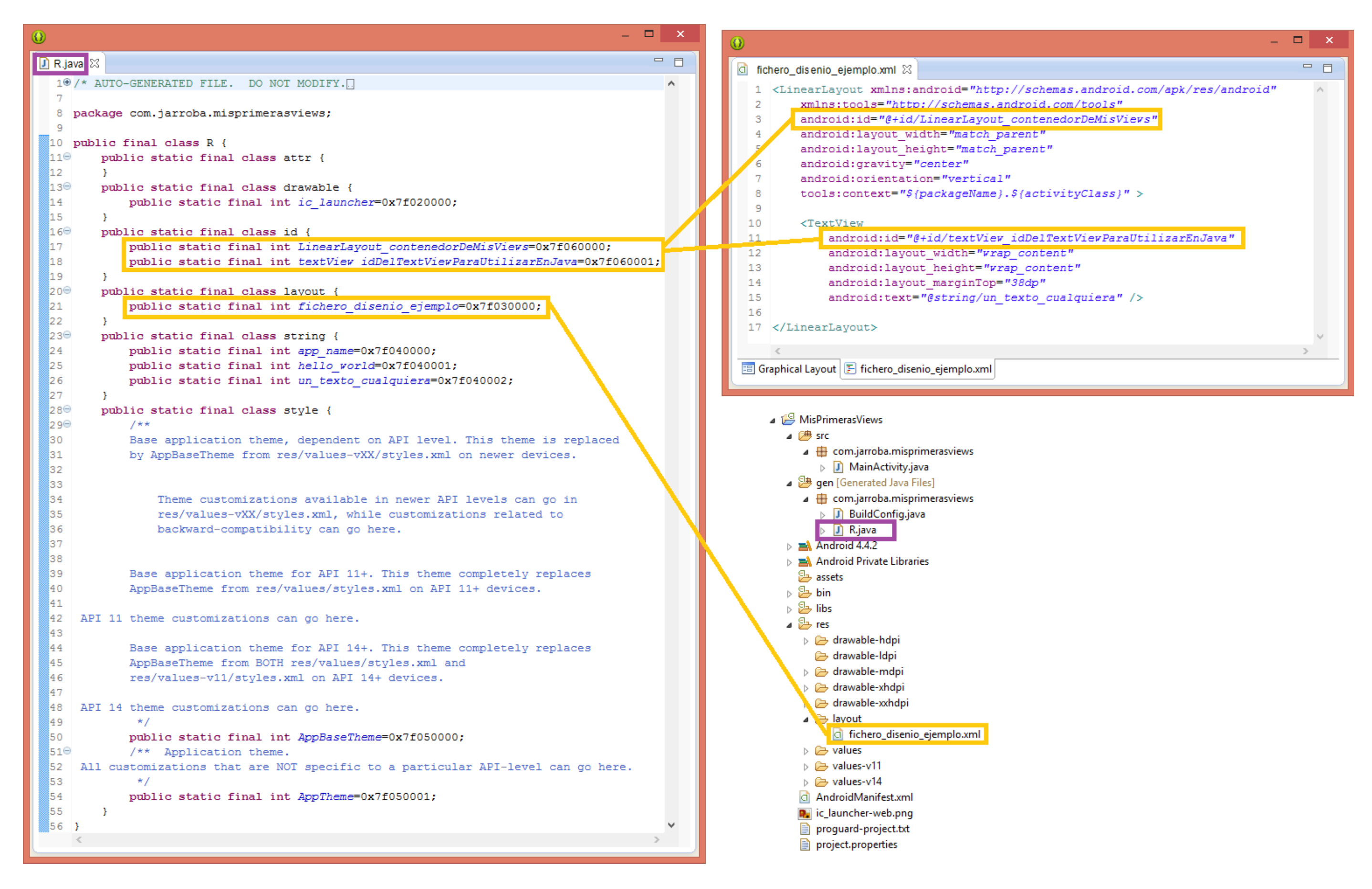
    <TextView
        android:id="@+id/textView_idDelTextViewParaUtilizarEnJava"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="38dp"
        android:text="@string/un_texto_cualquiera" />

</LinearLayout>
```



Has de hacerte la siguiente pregunta obligada ¿Cómo hago para poder utilizar desde Java los elementos que he creado en un fichero XML? Android nos lo facilita al crear automáticamente una clase intermedia llamada “R.java” que se encuentra dentro de la carpeta “gen”. Esta clase se actualizará de manera automática cada vez que guardemos un fichero XML de los que Android reconozca de los que están en la carpeta “res” (Por cierto, la R de “R.java” viene de la primera letra de la palabra recursos en inglé que es Resources, que casualidad que la carpeta “res” también). Este fichero “R.java” contendrá el nombre del id que le hemos puesto al elemento XML en el atributo “Android:id” y siempre que este empiece por “@+id/”.

Observa la siguiente imagen, fíjate que el ADT de Android agrupa en “R.java” los ids de las Views del diseño del XML dentro de una clase llamada “id” (Recuerda esto para cuando utilicemos findViewById()). Y que los ficheros de XML de la carpeta “layout” también están agrupados dentro de la clase “layout”.



A partir de ahora “R.java” no lo volveremos a abrir nunca y menos tocarlo (como he dicho anteriormente es el ADT de Android el que nos lo generará de manera automática; si lo manipulamos, lo más probable es que o bien nos empiece a dar errores el proyecto, o que el ADT lo regenere y perdamos todos nuestros cambios). Entender cómo se clasifican los identificadores dentro de “R.java” es fundamental para que la programación Android sea muy sencilla.

Ahora queremos llamarlo desde Java para encontrar el fichero de diseño en mediante el método setContentView(). Pues como utilizamos cualquier clase en Java empezamos buscándolo primero con el nombre de la clase más externa que es “R”; esta clase nos pide un archivo de diseño, con lo que seguiremos construyendo la referencia quedándonos “R.layout”; y ahora nos qeuda el nombre del fichero, formando nuestra referencia con “R.layout.fichero_disenio_ejemplo”.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.fichero_disenio_ejemplo);
}
```

Para encontrar una View la tenemos que encontrar por “id” con el método findViewById() de la Activity. Este método pide el identificador de recurso del “id” de la View de diseño. De manera análoga a la anterior, pero buscando una “id” la podremos encontrar con “R.id.XXXX”. Para encontrar la View de nuestro TextView usaremos en findViewById() el identificador de recurso “R.id.textView_idDelTextViewParaUtilizarEnJava”.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.fichero_disenio_ejemplo);

       View miView = findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);
}
```

¿Tengo que acordarme de todos los identificadores de recurso?

No hace falta. Un truco de Eclipse es la función de autocompletado. Al ir escribiendo nos aparecerán las diferentes posibilidades y como “R.java” es una clase Java, funciona exactamente igual que cualquier otra.

Si no aparecen las propuestas al ir escribiendo, posicionando el cursor de escritura donde queramos que aparezcan pulsamos la combinación de teclas [Ctrl] + [Espacio] y estás aparecerán.

```
*MainActivity.java

package com.jarroba.misprimerasviews;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fichero_disenio_ejemplo);

        View miView = findViewById(R.id.);
    }
}

LinearLayout_contenedorDeMisViews : int - R.id
textView_idDelTextViewParaUtilizarEnJava : int - R.id
class : Class<com.jarroba.misprimerasviews.R.id>
this
Press 'Ctrl+Space' to show Template Proposals
```

Todavía no podemos utilizar nuestro TextView, lee la siguiente pregunta.

¿Ya puedo utilizar mi View después de llamar al método findViewById()?

TextView, Button, RadioButton, CheckBox, LinearLayout, etc heredan todos de View. El método findViewById() devuelve una View. Para poder utilizar uno de ellos tenemos que realizar un cast al tipo apropiado.

El anterior ejemplo, si queremos usar nuestro TextView, tenemos que castearlo al tipo TextView de la siguiente manera (en este Eclipse nos pedirá importar “android.widget.TextView”):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.fichero_disenio_ejemplo);

      View miView = findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);

      TextView miTexto = (TextView) miView;
}
```

Ahora sí que lo podríamos utilizar el TextView para lo que queramos desde Java. Como cambiar el texto, modificar los colores, ocultarlo, añadirle animaciones, etc.

Para que en futuro sea más directo puedes simplificar lo anterior a esto:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.fichero_disenio_ejemplo);

      TextView miTexto = (TextView) findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);
}
```

Es importante saber que encontrar Views hay que hacerlo después de asociar el fichero de diseño completo a la Activity (es decir, después del método setContentView()). Y que la View que busquemos ha de estar en este fichero asociado con la Activity con la que trabajemos. De otra forma, dará error.

Se recomienda buscar las Views en el onCreate() y asignarlas a variables globales (en el ejemplo anterior son locales; recomiendo convertir a la variables de las Views a globales para poder llamarlas desde otras partes del ciclo de vida de la Activity).

¿Y si no me reconoce el “id” de la View porque no se actualiza el fichero “R.java”?

Remarco lo de que el fichero “R.java” se actualizará cuando guardemos los ficheros de recurso XML.

Este problema suele ser muy frustrante cuando empezamos a programar con Android. Y es tan fácil de corregir, con solo pulsar sobre el botón de “Save All” (guardar todo), para que se guarde nuestro diseño XML.

¿Y ahora como hago algo con la View?

Muy sencillo, ya solo tenemos que buscar entre todos los métodos que nos ofrece la propia View.

Por ejemplo cambiar el texto desde Java.

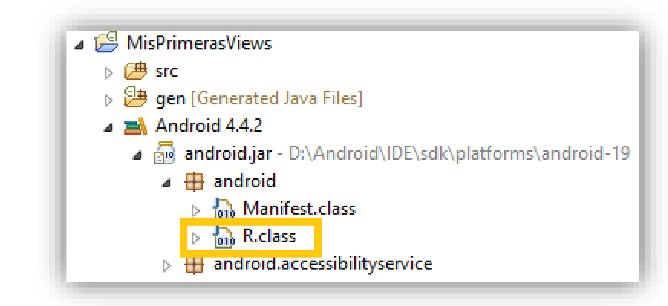
```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.fichero_disenio_ejemplo);

	TextView miTexto = (TextView) findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);

	miTexto.setText("El texto cambiado desde Java");
}
```

Para verlos todos con su descripción de lo que hacen, o nos vamos a la documentación de Android (http://developer.android.com/reference/packages.html), o como lo miramos en las sugerencias como se comentó en el anterior cuadro de nota azul que explica como mostrar las propuestas para autocompletar.

¿Android debe de tener recursos internos que pueda reutilizar?



Puede que nos guste algún recurso de Android que ya tenga definido. Para ello existe otro fichero “R.java” dentro de la biblioteca “android.jar” en el paquete “android”.

Imaginemos que queremos llamar a un color que ya tenga definida esta biblioteca. En vez de empezar por “R.color.XXX” que llamaría al “R.java” de la carpeta “gen”, es decir, el de nuestro proyecto, lo llamaríamos de este modo “Android.R.color.XXX”.

Por ejemplo, supongamos que queremos poner el texto de nuestro TextView de color verde luminoso que Android lo tiene llamado “holo_green_light”.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.fichero_disenio_ejemplo);

	TextView miTexto = (TextView) findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);

	int colorElegido = getResources().getColor(android.R.color.holo_green_light);
	miTexto.setTextColor(colorElegido);
}
```

Siempre que podamos, lo mejor es utilizar estos recursos, pues nos agilizará el trabajo y el proyecto ocupará menos que si tuviéramos que definirlos. Existen muchos recursos ya definidos por Android diferentes, desde ficheros de diseño XML hasta Strings, iremos viendo algunos a lo largo del libro.

¿Cómo encuentro desde Java una View para poder utilizarla en un Fragment?

Parecido a como se hace desde una Activity, pero con unos ligeros retoques.

En Fragment no existe el método setContentView(). Como vimos, el fichero de diseño XML se asocia a una Activity desde el método del ciclo de vida onCreateView().

Se pueden buscar las Views en el mismo método onCreateView(), justo después del inflado.

Para ello buscaremos la View mediante su id con el mismo método findViewById(). Para ello tenemos que utilizar la View que inflamos previamente, que en este ejemplo hemos llamado “rootView”.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fichero_disenio_ejemplo, container, false);

        TextView miTexto = (TextView) rootView.findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);

        return rootView;
}
```

Cabe mencionar que, la idea es que las variables de las Views que busquemos aquí las pongamos como globales. Por ello hay que tener cuidado en que parte del ciclo de vida utilizamos luego estas variables. Si te fijas en el ciclo de vida, si quisiéramos utilizar estas variables, por ejemplo, en el método onAttach() -como se ejecuta antes que el método onCreateView()- nos dará un error de puntero a null; debido a que no se habrá llamado todavía al método findViewById() que busca la View. Lo mismo pasaría si quisiéramos utilizar una View previamente buscada en el método onDetach(), que ya habría sido destruida la View con antelación, dando un null. Queriendo decir con esto, que hay que utilizar la View en los métodos de después del adjuntado, pero antes de la destrucción.

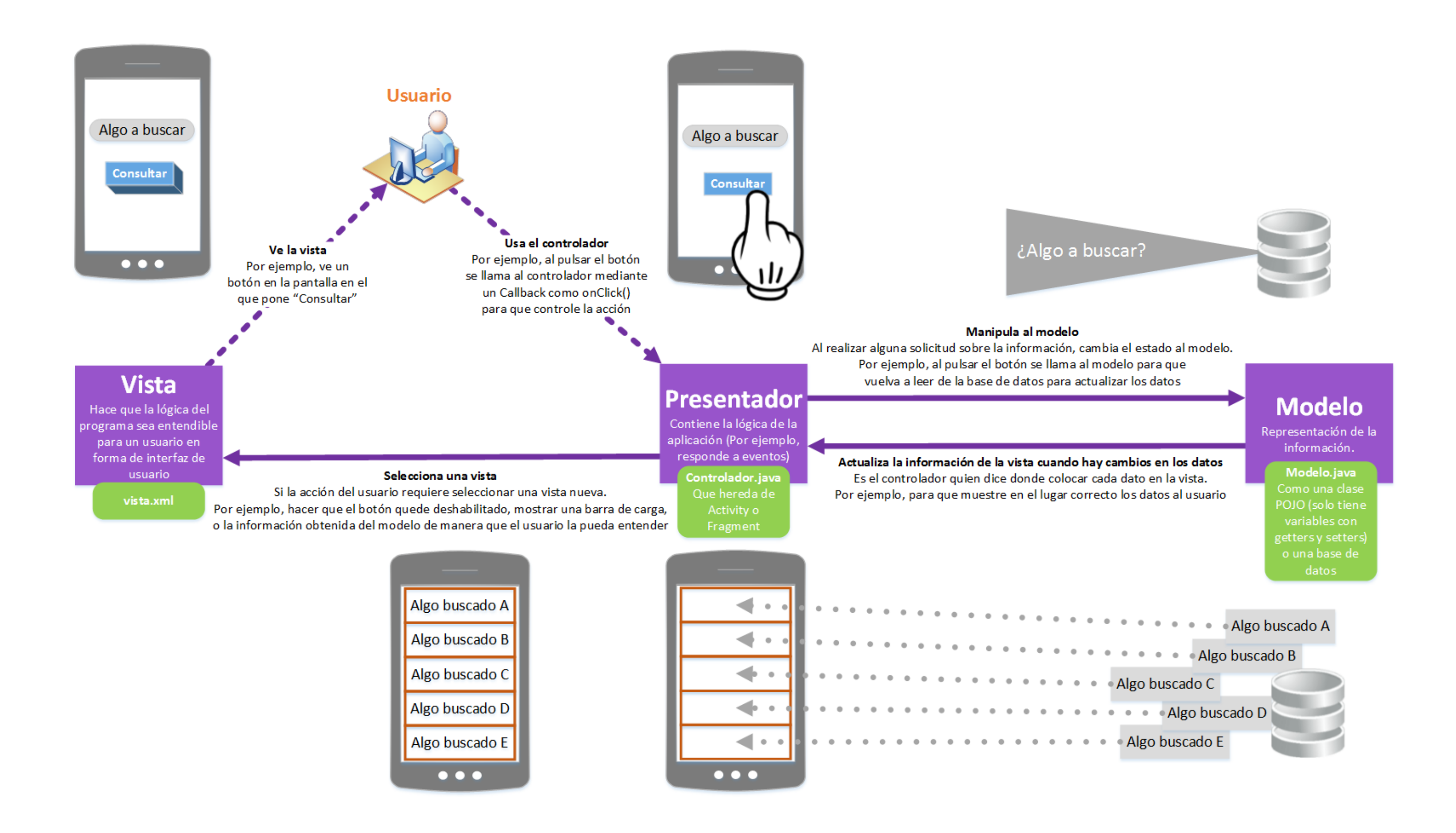
¿Qué patrón de diseño se recomienda seguir para trabajar con las interfaces de usuario en Android?

Podremos elegir el patrón de diseño que más nos convenga. Los más comunes son o Modelo-Vista-Controlador (MVC) o Modelo-Vista-Presentador (MVP).

Estos patrones de diseño son concretamente patrones arquitectónicos ya que ofrecen una arquitectura para la creación de cualquier proyecto Android, indicando como pueden ser relacionadas sus partes (vistas, lógicas, etc) y por tanto utilizados.

¿Y cuál me recomiendas utilizar?

Para Android recomiendo utilizar el patrón MVP. El patrón MVP funciona parecido al MVC (Modelo-Vista-Controlador), pero con la diferencia de no existir una relación directa entre el Modelo y la Vista. Toda comunicación ha de pasar necesariamente por el Presentador.



Aunque aquí recomendemos el patrón MVP por facilidad junto con la arquitectura de Android. Pero nada te prohíbe que utilices el MVC.

¿Cómo seguir al patrón MVP?

Aquí la lógica radica en que en Android solo se manejan las Vistas (los ficheros de diseño de la carpeta "res/layout") desde las Activities y los Fragmetns que actuarían de presentadores. Y el modelo de datos, por limpieza de código y reusabilidad se suele poner en ficheros aparte.

Ya hemos visto cómo utilizar las Views desde Android y Fragment. De este patrón MVP solo nos quedaría el Modelo. Lo veremos más adelante con el tratamiento de datos.

Para que te hagas una idea rápida, podríamos seguir una estructura de ficheros como en la imagen adyacente.

Referencias:

- http://es.wikipedia.org/wiki/Modelo_Vista_Controlador
- http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter
- http://stackoverflow.com/questions/4916209/which-design-patterns-are-used-on-android/6770903#6770903

Ejemplo de controlar las Views (por tanto diseños) desde Java

Lo que haremos

La Activity tendrá un diseño XML con un único texto inicial. Este texto lo cambiaremos por otro, le pondremos un fondo verde y lo rotaremos 45º. Haremos las cosas bien y los textos los crearemos directamente sobre el fichero "values/strings.xml"

Nada más cargar la aplicación nos mostrará esta imagen.



Ilustración 11 - En el instante de arrancar. Pasa tan rápido que es imperceptible para el ojo humano

Aunque nunca la llegaremos a ver pues desde Java cambia muy rápido a la siguiente.

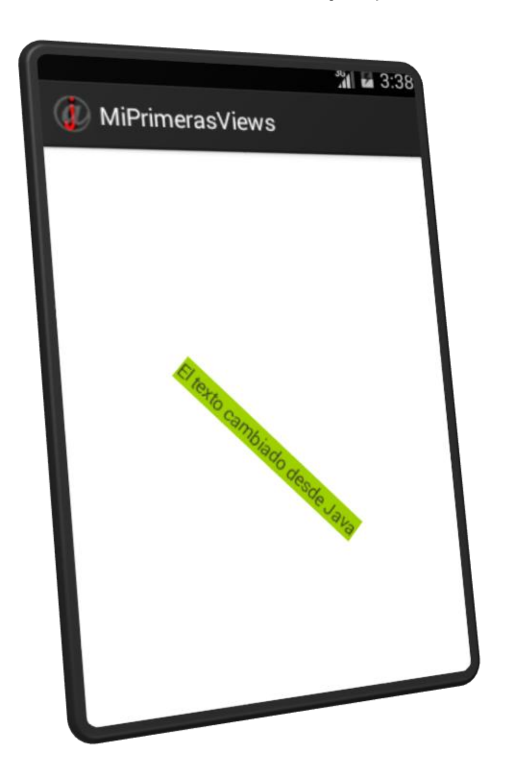
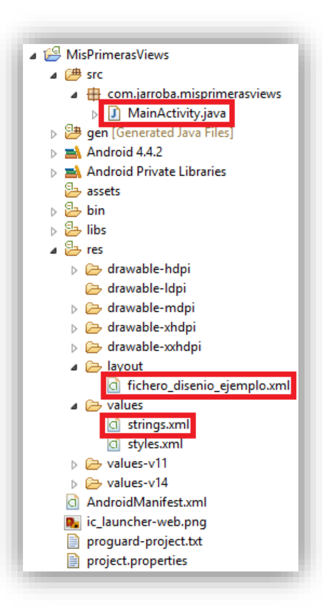


Ilustración 12 - Después de aplicar los cambios desde Java



MainActivity.java

```java
public class MainActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.fichero_disenio_ejemplo);

		TextView miTexto = (TextView) findViewById(R.id.textView_idDelTextViewParaUtilizarEnJava);

		miTexto.setText(R.string.el_texto_cambiado_desde_java);

		int colorElegido = getResources().getColor(android.R.color.holo_green_light);
		miTexto.setBackgroundColor(colorElegido);

		miTexto.setRotation(45.0f);
	}
}
```

res/layout/fichero_disenio_ejemplo.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout_contenedorDeMisViews"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="${packageName}.${activityClass}" >

    <TextView
        android:id="@+id/textView_idDelTextViewParaUtilizarEnJava"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="38dp"
        android:text="@string/un_texto_cualquiera" />

</LinearLayout>
```

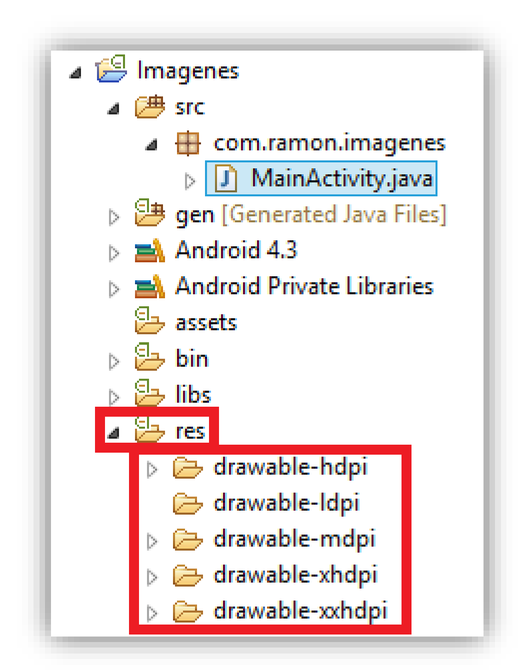
res/layout/fichero_disenio_ejemplo.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">MiPrimerasViews</string>
    <string name="un_texto_cualquiera">Un texto cualquiera</string>
    <string name="el_texto_cambiado_desde_java">El texto cambiado desde Java</string>

</resources>
```

Drawable



¿Qué es?

Es un recurso que puede ser dibujado en pantalla. Se encuentran en Se encuentra en la carpeta “res”, dentro de alguna carpeta “drawable”

Si creo una imagen de tipo png con un programa de edición de imágenes como Gimp o Photoshop ¿Cuántas imágenes tengo que hacer?

Funcionar funciona con una, ya que Android buscará la imagen a la que se haga referencia en cualquier carpeta “drawable” que más se acerque a las especificaciones de pantalla del dispositivo.

Para hacer las cosas bien habría que hacer una para cada carpeta “drawable”, atendiendo a las densidades. Es mejor así, ya que cambiar el tamaño de una imagen en tiempo de ejecución consume muchos recursos que podría notar el usuario (consumo de batería, agotamiento de la memoria, lentitud de respuesta). Además, las funciones de escalado eliminan píxeles de la imagen, o añaden píxeles inventados; por lo que genera una imagen de mala calidad o borrosa. Lo mejor es tener una imagen para cada carpeta de las densidades recomendadas (una excepción es la carpeta “drawable-ldpi”, a la que no hace falta poner imagen, ya que cambio de tamaño hacia abajo y debido a su pequeño tamaño, no se notan los efectos negativos antes citados)

¿Sólo se guardan imágenes png, jpg o gif en estas carpetas?

No solo esas, sino que podemos guardar ciertos ficheros XML que definan imágenes, como veremos a continuación.

¿Las imágenes que guarde en las carpetas drawable permanecerán inalteradas?

Toda imagen guardada en la carpeta “res/drawable” será optimizada y comprimida. Si se quiere evitar la optimización hay que colocarla en la carpeta “res/raw”.

¿Qué tipos de imágenes se pueden guardar en las carpetas Drawable?

- **Bitmap**: imagen con extensión png, jpg o gif
- **9-path**: png con regiones que se pueden estirar según el tamaño del contenido sin deformar ciertas áreas
- **Layer List**: XML que controla una colección de Drawables a modo de capas
- **State List**: XML que referencia diferentes ficheros Bitmaps para diferentes estados

- **Level List**: XML que define un drawable que maneja un número de Drawables alternativos
- **Transition Drawable**: XML que define un drawable que puede fundirse entre dos drawables
- **Inset Drawable**: XML que define un drawable que inserta en otro drawable a una distancia específica. Útil para repetir una imagen varias veces en un fondo
- **Clip Drawable**: XML que define un drawable que se engancha a otro, basado en su valor de nivel actual
- **Scale Drawable**: XML que define un drawable que cambia de tamaño de otro, basado en su valor de nivel actual
- **Shap Drawable**: XML que define una figura geométrica, incluido color y gradientes

Referencias:

- http://developer.android.com/guide/topics/resources/drawable-resource.html
- http://developer.android.com/guide/topics/graphics/2d-graphics.html

Bitmap (Imagen o mapa de bits)



¿Qué es?

Es una imagen en formato compatible como png, jpg o gif. Se recomienda png.

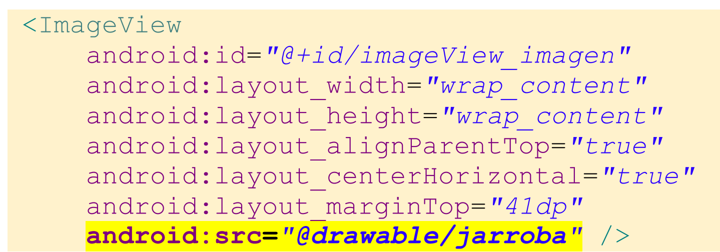
¿Para qué se usa?

Para cualquier texto que no vaya a ser modificado durante la ejecución de la aplicación

¿Cómo se obtiene?

Se puede usar en cualquier View que permita imágenes (propiedad "src") o fondos (propiedad "background"), como ImageView, Button, TextView, etc

- En los recursos XML con: **@drawable/imagen**



```
<ImageView
    android:id="@+id/imageView_imagen"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="41dp"
    android:src="@drawable/jarroba" />
```

- En Java con: **getResources().getDrawable(R.drawable.imagen)**

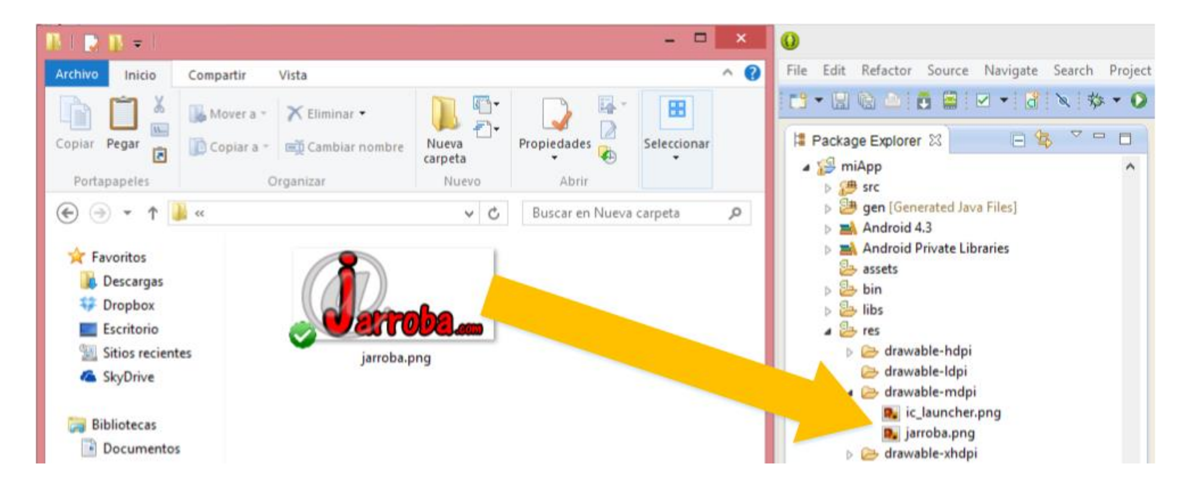
```
Drawable miImagen = getResources().getDrawable(R.drawable.jarroba);

ImageView iv = (ImageView) findViewById(R.id.imageView_imagen);

iv.setImageDrawable(miImagen);
```

¿Cómo añado una imagen a alguna carpeta "drawable" del proyecto?

Vale con arrastrarla a la carpeta "drawable" adecuada. Solo asegúrate de una cosa, que el nombre de la imagen esté bien formado (sino dará errores y no se autogenerarán los identificadores en el fichero "R.java"). Es decir, que las letras sean minúsculas, sin caracteres raros (solo se permiten guiones bajos "_" y el símbolo del dólar "$"); y que la extensión sea correcta como ".png"



9-patch (Nueve partes)

¿Qué es?

Es una imagen en formato png con regiones que se pueden estirar según tamaño del contenido, sin deformar las esquinas. Se llama nueve partes (9-patch) porque dividiremos la imagen en nueve partes, unas que se podrán estirar y otras que no.

¿Para qué se usa?

Normalmente para botones, en los que va a ir un texto de tamaño desconocido dentro, y queremos que esté siempre dentro del botón, con lo que la imagen se ajustará sin deformar las partes que no queremos que se deformen.

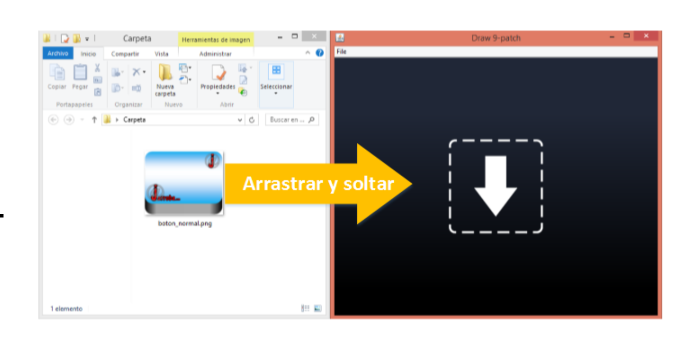
¿Cómo se obtiene?

Tanto en Java como en XML de la misma manera que en Bitmap

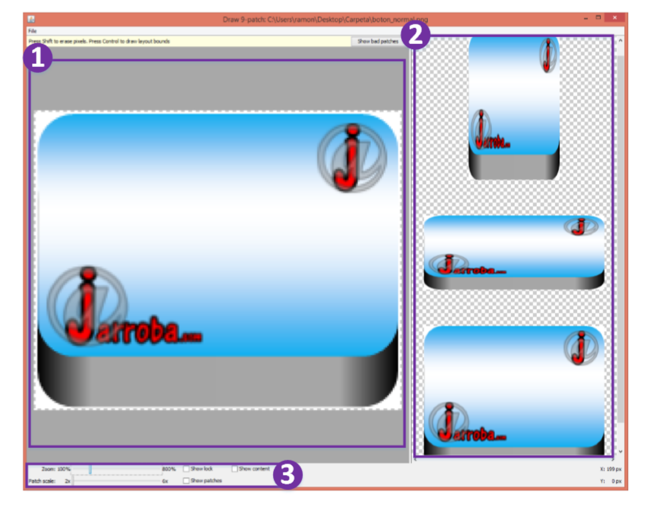
¿Hay alguna manera sencilla de crear una imagen 9-pach?

Sí, con el editor de imágenes 9-patch que trae el SDK de Android y que se encuentra en "**\IDE\sdk\tools**", aquí ejecutaremos el "**draw9patch.bat**".

El programa se inicia pidiéndonos una imagen. Arrastraremos la imagen que queramos convertir a 9-patch.



¿En qué consta el editor de imágenes 9-pach?



Después de arrastrar a la imagen, cargará y podremos editar las nueve partes de una manera muy sencilla. El editor se divide en:

1.**Editor**: Veremos nuestra imagen en grande a la derecha, donde podremos editar las partes

2.**Vista previa**: Aparece también en la vertical nuestra imagen tres veces, esta parte es solo una vista previa de cómo quedará (vista previa de si se estirara verticalmente, horizontalmente, o en los dos).

3.**Controles**: Los controles de nuestro editor (están descrita su funcionalidad en la siguiente pregunta)

¿Para qué sirven los controles del editor de imágenes 9-pach?

- **Zoom**: Ajusta el nivel del zoom en el área de dibujo
- **Patch scale**: Ajusta el nivel del zoom en el área de la vista previa
- **Show lock**: Visualiza las áreas no dibujables al pasar el cursor del ratón por encima
- **Show patches**: Mostrar las partes que se podrán estirar del área de la imagen (La parte de color rosa se podrá estirar tanto a lo ancho como a lo alto, las partes verdes de izquierda y derecha tan solo se podrán

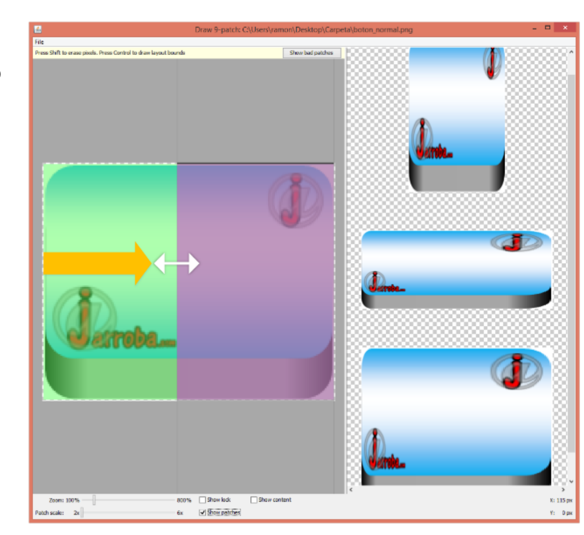
estirar a lo alto, las partes verdes de arriba y abajo solo se pondrán estirar a lo ancho, y las partes sin área de color no se estirarán).

- **Show content**: Mostrar en morado el área donde se permitirá el contenido en la vista previa
- **Show bad patches**: Añade un borde rojo alrededor de las áreas de los parches que podrían producir defectos en la imagen cuando se estire. Se verá siempre bien si no existen áreas rojas

¿Cómo se utiliza el editor de imágenes 9-pach?



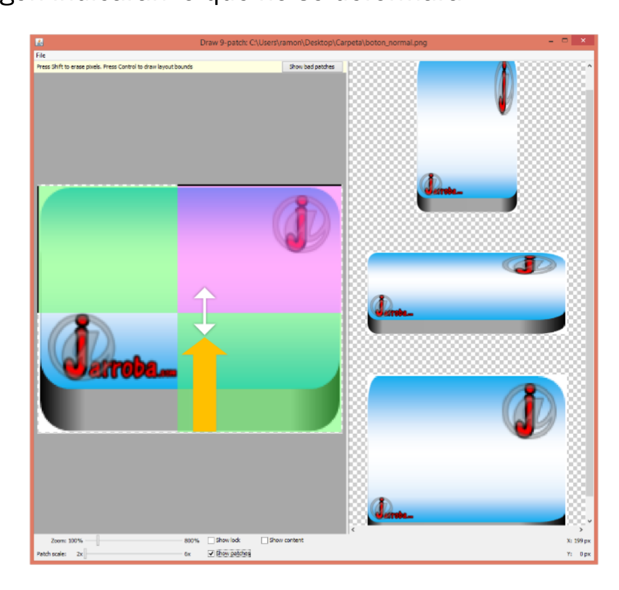
Antes de nada, para ver con mayor claridad cómo quedarán las nueve partes, recomiendo seleccionar la casilla "**Show patches**".



Ahora, si nos vamos al editor con el cursor del ratón y lo ponemos sobre uno de los cuatro lados de la imagen y tiramos hacia dentro estaremos indicando lo que no queremos que se deforme de nuestra imagen. Por ejemplo, si queremos estrechar las partes verticales de la imagen, bastará con seleccionar desde el borde izquierdo o derecho de la imagen y arrastrar a la posición que queramos que no sea modificada. En la siguiente imagen de ejemplo, queremos que no se deforme ni las esquinas de la izquierda, ni la imagen que hay en ella (el logotipo de Jarroba); pues arrastramos para cubrir las esquinas con el logo soltamos.

De este modo veremos cómo se nos habrá quedado una parte rosa y otra verde. Cuando terminemos de definir las nueve partes de la imagen tendremos las siguientes con sus significados:

- **1 Parte rosa**: se podrá estirar tanto a lo ancho como a lo alto,
- **4 Partes verdes**: las partes verdes de izquierda y derecha tan solo se podrán estirar a lo alto; las de arriba y abajo solo se pondrán estirar a lo ancho, y las partes sin área de color no se estirarán
- **4 Partes sin color**: estas zonas en las esquinas de la imagen indicarán lo que no se deformará

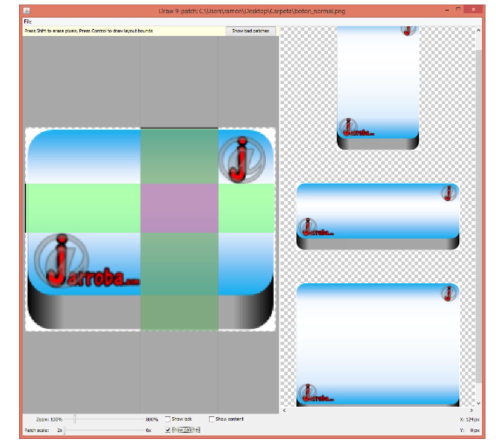


Por lo que la imagen deformará el logotipo de Jarroba en la horizontal (en la imagen superior, si miramos primera vista previa, el logo queda aplastado horizontalmente).

Lo arreglamos al desplazar el borde inferior de la imagen. Por lo que veremos la parte sin sobrexponer ni el rosa ni el verde, lo que indicará que no se deformará esa parte de la imagen. Echa un vistazo a la siguiente captura y en especial a su vista previa, como no se deforma el logotipo de Jarroba.

Tan solo quedarán por definir desde el borde superior y derecho de la imagen.

Para terminar guardaremos la imagen en "File" y en "Save 9-patch..."



¿Por qué le salen rayas negras a mi imagen 9-patch en los bordes?



Mientras la editábamos y al abrir la imagen 9-patch podremos ver como el editor nos ha generado unas líneas negras a los bordes de la imagen. Estas líneas indicarán a Android las partes que no queremos que deforme. Podríamos haber hecho estas líneas negras con un editor de imágenes, pero habría que tener cuidado de hacerlas sobre un píxel de fondo transparente y cuando guardáramos la imagen con extensión ".9.png" (todo esto nos lo ahorramos si utilizamos el editor 9-patch incluido en el SDK de Android).

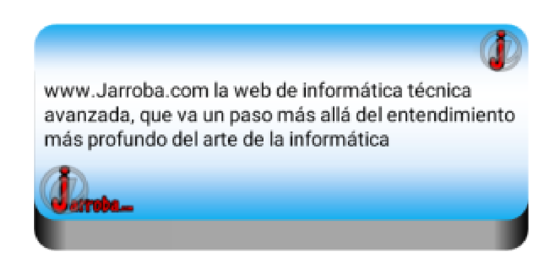
¿Cómo uso una imagen con extensión ".9.png"?

Exactamente igual que las imágenes bitmap.

Un ejemplo rápido es en un diseño en XML, añadir un "TextView", y poner de fondo el "android:background".

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/boton_normal"
    android:paddingBottom="90dp"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:paddingTop="50dp"
    android:text="www.Jarroba.com la web de informática técnica
avanzada, que va un paso más allá del entendimiento más profundo
del arte de la informática "
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

Le pondremos un texto largo para ver en el mismo editor que se deforma de la imagen y que se mantiene sin deformarse (le aplico un padding para que el texto quede colocado en la imagen). En este ejemplo veremos que las esquinas y los logos no se deformarán, pero lo que es el cuerpo sí, justo como queríamos.



Jugando con el padding podremos ver qué ocurre cuando el texto se recoloca dentro del TextView. El siguiente ejemplo le damos más padding por la izquierda y la derecha para que el texto se coloque en una alta columna:



Referencias:

- http://developer.android.com/tools/help/draw9patch.html

State List (Lista de estados)

¿Qué es?

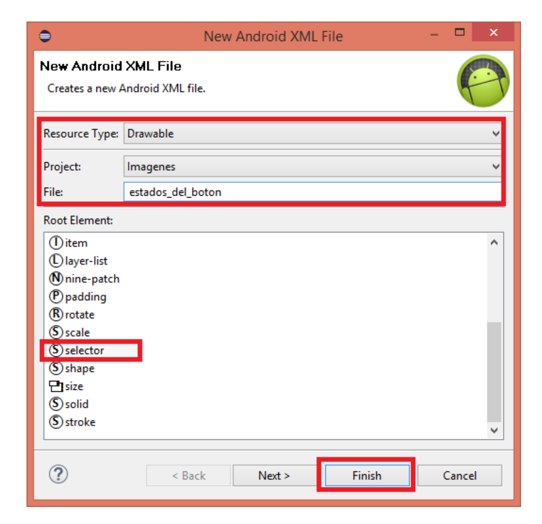
Es un XML que referencia diferentes ficheros Bitmaps para diferentes estados definidos en una lista

¿Para qué se usa?

Por ejemplo, para cada estado del botón: pulsado, enfocado, reposo, etc

¿Cómo se define un XML de State List?

Se crea un nuevo XML en la carpeta “**drawable**” de tipo “**selector**”



En este fichero añadiremos elementos (<ítem/>). Cada uno tendrá dos atributos, uno que diferenciará el estado y otro que indicará la imagen que queremos que se le muestre al usuario cuando esté la View en dicho estado. En el siguiente código hemos definido tres estados (aunque hay muchos más) y para cada uno una imagen:



- Botón pulsado: para cuando el usuario pulse el botón



- Botón enfocado: para cuando el usuario seleccione el botón con un cursor



- Botón en cualquier otro estado: lo usaremos para cuando el botón esté en reposo

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/boton_pulsado" /> <!-- pulsado -->
    <item android:state_focused="true"
          android:drawable="@drawable/boton_enfocado" /> <!-- enfocado -->
    <item android:drawable="@drawable/boton_normal" /> <!-- por defecto, el botón
normal en estado de reposo -->
</selector>
```

¿Cómo uso una imagen State List?

Exactamente igual que un Bitmap.

Un ejemplo rápido es en un diseño en XML, añadir un “Button”, y poner de fondo el “android:background” el XML del State List que hemos definido.

```
<Button
    android:id="@+id/button_miboton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/fichero_state_list"
    android:text="Un boton" />
```

Shap Drawable (Dibujable de forma)

¿Qué es?

Es un XML que define una figura geométrica, incluido colores y gradientes. Se podría diseñar una imagen normal con una herramienta de dibujo como Gimp o Photoshop; sin embargo, al definirlo en XML no perdería calidad y se dibujaría siempre la imagen completa. Un ejemplo, si queremos que la imagen se componga de un borde de línea discontinua, y cada línea no tenga más de unos píxeles de tamaño; si lo hiciéramos con un png la línea punteada deformaría sus trozos estirándolos, al diseñar una forma conseguimos que todas las líneas sean pequeñas

¿Para qué se usa?

Se suele utilizar para crear fondos de Views; por ejemplo, para el fondo de un botón o de un EditText personalizado. O para líneas de separación de listados por ejemplo. Aunque también para lo que queramos.

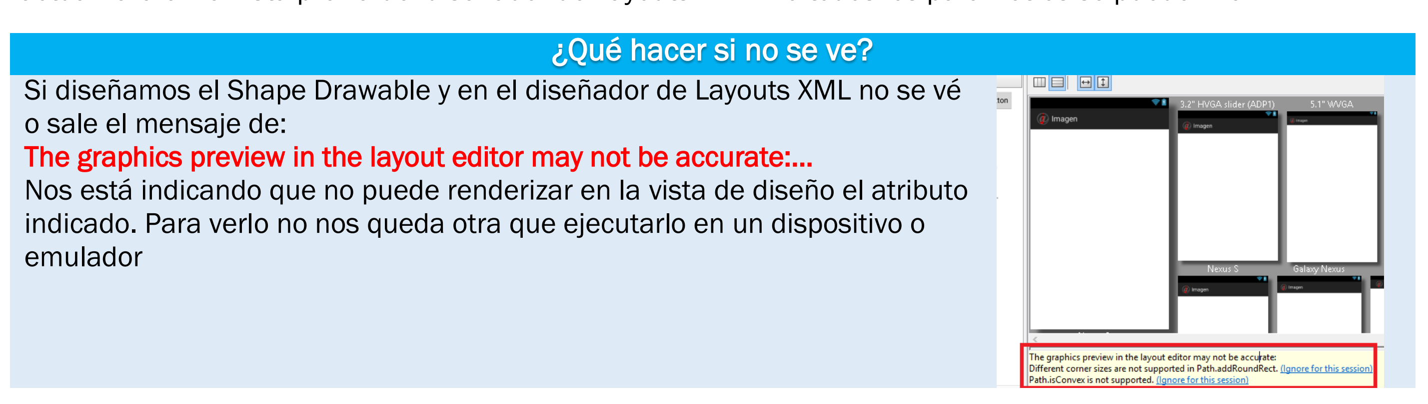
No podremos crear cualquier dibujo que queramos, solo las formas de línea, de rectángulo (incluye cuadrado), de óvalo (incluye círculo) y de anillo

¿Cómo determino su tamaño?

Al asignarse a una View (por ejemplo como fondo), estas figuras es expandirán horizontal y verticalmente hasta ocupar toda la View (salvo la línea, que se extiende solo a lo largo)

¿Puedo ver la Shape Drawable mientras lo diseño?

Sí, si lo asignas en algún diseño en XML, como en el fondo de un botón. Cada vez que guardes la forma se actualizará en la vista previa del diseñador de Layouts XML. No todos los parámetros se pueden ver



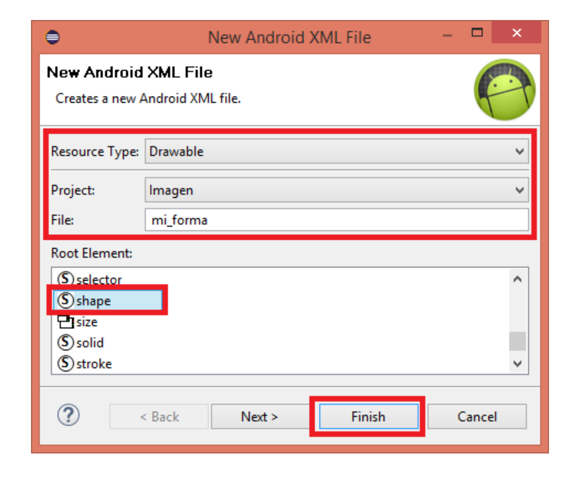
¿Qué hacer si no se ve?

Si diseñamos el Shape Drawable y en el diseñador de Layouts XML no se vé o sale el mensaje de:
The graphics preview in the layout editor may not be accurate:...
Nos está indicando que no puede renderizar en la vista de diseño el atributo indicado. Para verlo no nos queda otra que ejecutarlo en un dispositivo o emulador

¿Cómo se define un XML de State List?

Se crea un nuevo XML en la carpeta "**drawable**" de tipo "**shape**"

Veremos ejemplos de cómo se define en XML en la siguiente pregunta:



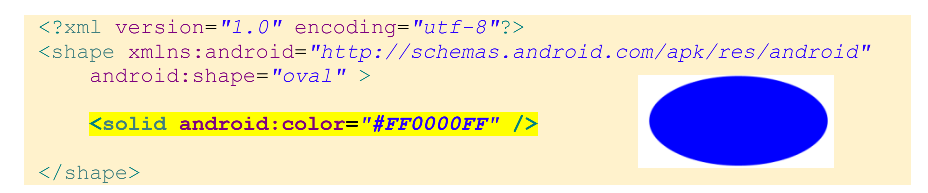
¿Qué se puede definir en una Shape Drawable?

Normalmente se utiliza para crear formas como líneas, rectángulos (con ello cuadrados), óvalos (con esto círculos) o anillos. Primero definiremos la figura que dibujaremos entre:

Forma	Ejemplo	Información
Rectángulo (rectángulo)		Jugando con los valores se consigue la forma de cuadrado
Oval (óvalo)		Jugando con los valores se consigue la forma de círculo
Line (línea)		No tiene cuerpo, es todo borde (requiere usar <stroke>)
Ring (Anillo)		Cabe notar que el cuerpo tiene forma de anillo (el centro de la figura está vacía). Y tiene dos bordes: uno interior y otro exterior

Se pueden asignar los siguientes parámetros (a continuación de cada uno, un ejemplo en cada cuadro amarillo):

- **Solid** (Sólido): Un color sólido para rellenar el cuerpo de la figura



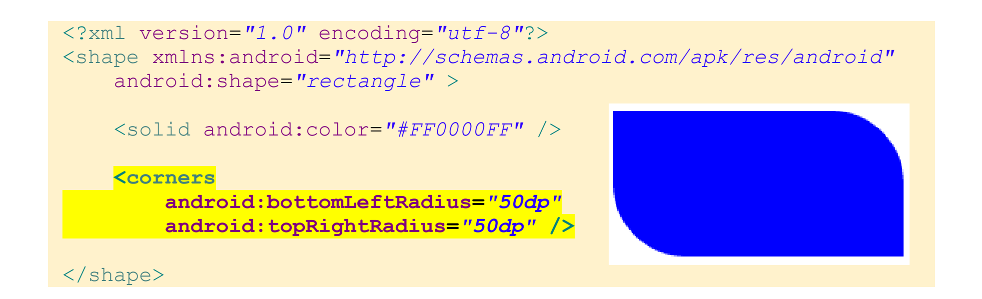
```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval" >

    <solid android:color="#FF0000FF" />

</shape>
```

- **Corner** (Esquina): básicamente es redondear la esquina de un rectángulo. Todas las medidas se definen en dp, siendo 0dp sin esquinas redondeadas.

Atributo	Definición	Ejemplo	Valor	Información
radius	radio de todas las esquinas		Desde 0dp	
topLeftRadius	radio superior izquierda		Desde 0dp	Para visualizar requiere la ejecución desde un dispositivo o emulador
topRightRadius	radio superior derecha		Desde 0dp	Para visualizar requiere la ejecución desde un dispositivo o emulador
bottomLeftRadius	radio inferior izquierda		Desde 0dp	Para visualizar requiere la ejecución desde un dispositivo o emulador
bottomRightRadius	radio inferior derecha		Desde 0dp	Para visualizar requiere la ejecución desde un dispositivo o emulador



```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

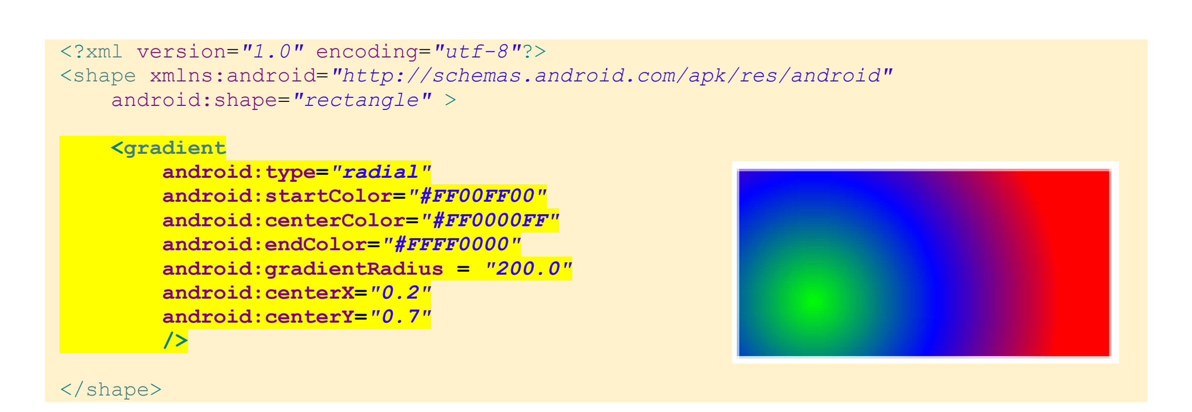
    <solid android:color="#FF0000FF" />

    <corners
        android:bottomLeftRadius="50dp"
        android:topRightRadius="50dp" />

</shape>
```

- **Gradient** (Gradiente o degradado de colores):

Atributo	Definición	Ejemplo	Valor	Información
startColor	Color inicial		Color #AARRGGBB o recurso de color	
centerColor	Color central		Color #AARRGGBB o recurso de color	
endColor	Color final		Color #AARRGGBB o recurso de color	
angle	Ángulo de inclinación del degradado		De 0 a 359 grados	
centerX	Desplazamiento horizontal del centro	0.0 0.5 1.0	0.0 (izquierda) a 1.0 (derecha)	También se puede aplicar para desplazar al tipo linear horizontalmente
centerY	Desplazamiento vertical del centro	0.0 0.5 1.0	0.0 (arriba) a 1.0 (abajo)	También se puede aplicar para desplazar al tipo linear verticalmente
gradientRadius	Radio del gradiente para el tipo radial	0.0	Desde 0.0 (todo concentrado en el centro)	Es obligatorio su uso en el tipo radial. En el resto de tipos no se aplica
type	Tipo del gradiente	Linear: Radial: Sweep:	Puede ser: **linear:** gradiente lineal **radial:** gradiente radial cuyo color central es el color inicial **sweep:** abanico de colores, empieza en un color y termina en un giro pegado al primero	



```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <gradient
        android:type="radial"
        android:startColor="#FF00FF00"
        android:centerColor="#FF0000FF"
        android:endColor="#FFFF0000"
        android:gradientRadius = "200.0"
        android:centerX="0.2"
        android:centerY="0.7"
        />

</shape>
```

- **Stroke** (Línea de borde):

Atributo	Definición	Ejemplo	Valor	Información
width	Anchura del borde		Desde 0dp	
color	Color del borde		Color #AARRGGBB o recurso de color	
dashWidth	Anchura de cada guión		Desde 0dp	Utilizar junto al atributo “dashGap”
dashGap	Distancia entre guiones		Desde 0dp	Utilizar junto al atributo “dashWidth”



```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <solid android:color="#FF00FF00" />

    <stroke
        android:width="5dp"
        android:color="#FFFF0000"
        android:dashGap="4dp"
        android:dashWidth="8dp"
        />

</shape>
```

- **Padding** (Relleno): desplaza al contenido que pongamos sobre la Shape Drawable

Atributo	Definición	Ejemplo	Valor
left	Relleno izquierdo		Desde 0dp
right	Relleno derecho		Desde 0dp

Atributo	Definición	Ejemplo	Valor
top	Relleno superior		Desde 0dp
bottom	Relleno inferior		Desde 0dp



```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <solid android:color="#FFFF0000" />

    <padding
        android:left="40dp"
        android:top="50dp"
        />

</shape>
```


- **Size** (Tamaño): No se suele usar, ya que el tamaño lo determina la View donde se pone el Shape Drawable. Si quieres que se aplique este tamaño en una ImageView tendrás que poner su atributo Android:scaleType = “center”

Atributo	Definición	Ejemplo	Valor
height	Altura		Desde 0dp

Atributo	Definición	Ejemplo	Valor
width	Anchura		Desde 0dp

Ejemplo de imágenes

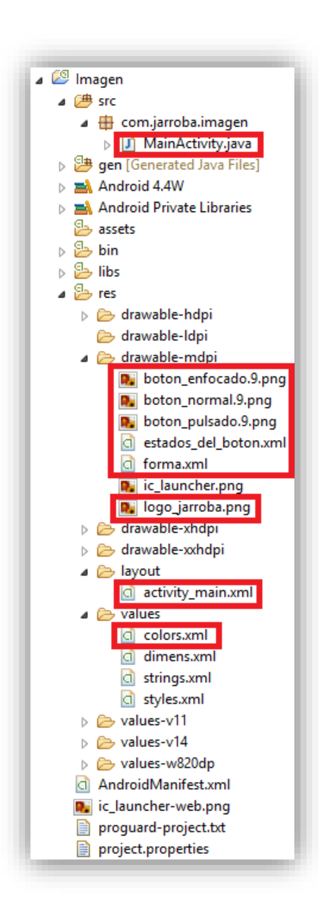
Lo que haremos

- Pondremos una **imagen personalizada**: para ello pondremos una imagen de nuestro logo en las carpetas drawables (por facilitar el ejemplo, hemos puesto todas las imágenes dentro de la carpeta "drawable-mdpi", aunque estaría mejor hacer una imagen para cada densidad). Luego se la asignaremos en el diseño XML al atributo "android:src" de una "ImageView".
- **Texto con fondo personalizado** definiéndolo desde XML en una forma: crearemos una "**Shape**" con un degradado de colores y un borde, que pondremos en el diseño XML de un "TextView" en el atributo "Android:background" de un "TextView". Además le pondremos un color al texto desde XML.
- **Botón de estados personalizados**: haremos una "**State List**": con tres estados de pulsado, enfocado y en reposo. Las imágenes de cada uno de los estados serán de tipo 9-path. Estos estados estarán asignados directamente a un "Button". Y a este botón, el texto le cambiaremos el color desde Java.



Nota para cambiar el enfoque: para enfocar el botón con el emulador basta con pulsar la tecla "Tabular" del teclado hasta posicionarnos encima del botón. Para un dispositivo, necesitamos uno con teclas de cursor, ya no se suelen utilizar mucho, por lo que puede que el enfoque quede obsoleto.

Nota sobre los Strings: Estaría mejor utilizar la carpeta "strings.xml" para guardar los textos que necesitemos (para simplificar el ejemplo no lo utilizo, pero debes usarlo para una mejor programación). Otra cosa más, utilizo "\n" en un "String" para indicar a Java que ahí quiero un salto de línea.

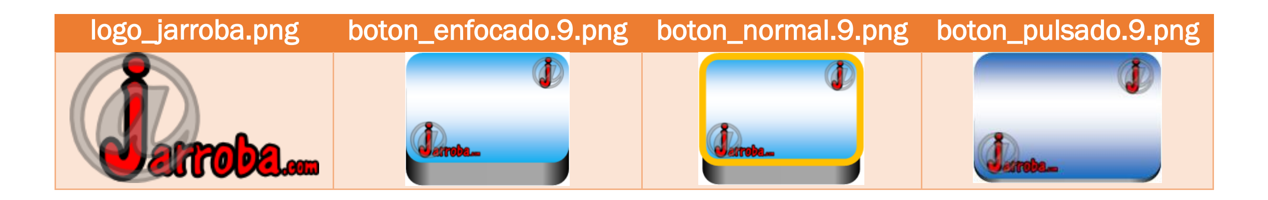


MainActivity.java

```java
public class MainActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                int colorElegido = getResources().getColor(R.color.Azul);

                TextView tv = (TextView) findViewById(R.id.textView_a_colorear);
                tv.setTextColor(colorElegido);
        }
}
```



res/drawable-mdpi/estados_del_boton.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/boton_pulsado" /> <!-- presionado -->
    <item android:state_focused="true"
          android:drawable="@drawable/boton_enfocado" /> <!-- enfocado -->
    <item android:state_hovered="true"
          android:drawable="@drawable/boton_enfocado" /> <!-- cursor encima -->
    <item android:drawable="@drawable/boton_normal" /> <!-- por defecto, el botón normal en
estado de reposo -->
</selector>
```

res/drawable-mdpi/forma.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="@color/Dorado"
        android:endColor="@color/Verde"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
    <stroke
        android:width="5dp"
        android:color="@color/Rosa"
        android:dashWidth="2dp"
        android:dashGap="2dp" />
</shape>
```

res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.jarroba.imagen.MainActivity" >

    <ImageView
        android:id="@+id/imageView_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/logo_jarroba"
        android:contentDescription="Descripción de la imágen" />

    <TextView
        android:id="@+id/textView_a_colorear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="30dp"
        android:background="@drawable/forma"
        android:text="Un texto cualquiera\nwww.Jarroba.com"
        android:textAppearance="?android:attr/textAppearanceMedium" />

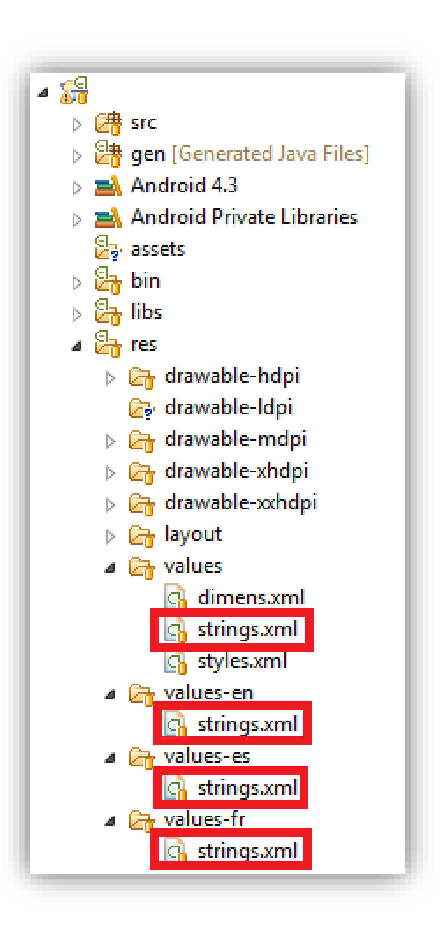
    <Button
        android:id="@+id/button_miboton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/estados_del_boton"
        android:text="Púlsame"
        android:textColor="@color/Morado" />

</LinearLayout>
```

res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="Verde">#FF00FF00</color>
    <color name="Azul">#FF0000FF</color>
    <color name="Morado">#FFC54B8C</color>
    <color name="Dorado">#FFFFD700</color>
    <color name="Rosa">#FFFF00FF</color>
</resources>
```

Strings (Textos)



¿Qué es un String inmutable?

Una cadena de texto se guarda en un objeto de tipo String. Que representa un array de char[] de valores codificados en UTF-16 (Unicode). Un String es un objeto inmutable (una vez creado su estado no puede ser cambiado)

¿Está bien poner los Strings directamente escritos (Hardcoded) en Java o en los XML?

En Android es recomendable tratarlos como recursos. En la carpeta "values".

¿Se puede tener el mismo texto traducido a diferentes idiomas?

Sí, se puede tener el mismo texto pero en diferentes idiomas. Como vimos, para ello tendremos que crear una carpeta "values" con un sufijo de idioma; por ejemplo: "values-fr"

¿Sólo se pueden tratar textos simples?

Existen tres tipos de recursos para tratar las cadenas de texto: String, String Array y Quantity Strings

Referencias:

- http://developer.android.com/reference/java/lang/String.html
- http://developer.android.com/guide/topics/resources/string-resource.html

String (String simple)

¿Qué es?

Una simple cadena de texto

res/values/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="escriba_para_guardar_">Escriba para guardar:</string>

</resources>
```

¿Para qué se usa?

Para cualquier texto que no vaya a ser modificado durante la ejecución de la aplicación

¿Cómo se obtiene?

- En los recursos XML con: **@string/nombre_string**

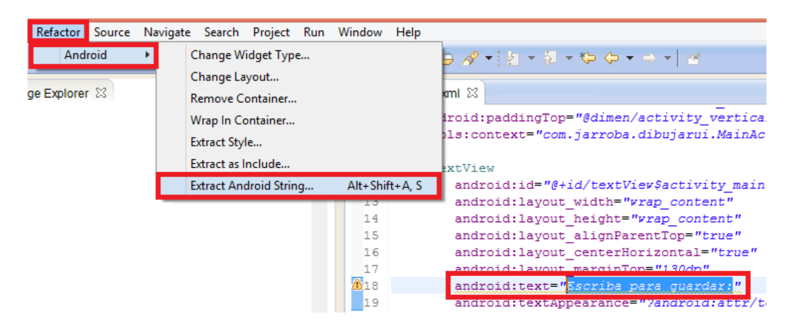
```
<TextView
        android:id="@+id/textView$activity_main$EscribirParaGuardar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="130dp"
        android:text="@string/escriba_para_guardar_"
        android:textAppearance="?android:attr/textAppearanceLarge" />
```

- En Java con: **getString(R.string.nombre_string)**

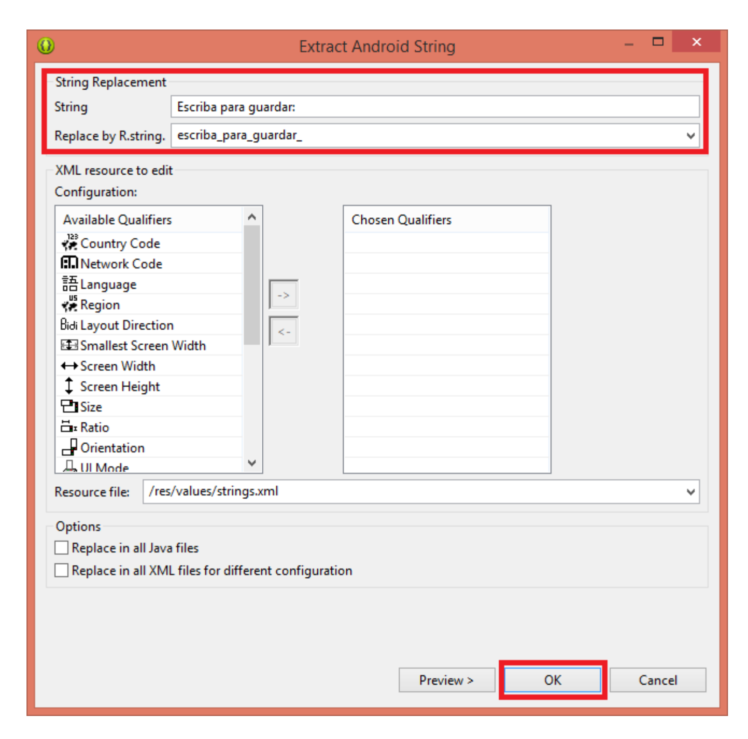
```
String textoDelBoton = getString(R.string.boton_guardar);
```

¿Existe alguna forma de extraer los Strings de los recursos XML, de manera fácil y automática?

Sí, es muy sencillo. Desde un recurso XML seleccionamos solo el texto que queramos extraer. Entonces pulsamos en la barra de herramientas en “Refactor/Android/Extract Android String…”.



Se nos abrirá una nueva ventana para que configuremos la nueva propiedad del recurso que vayamos a extraer. Aquí se nos rellenarán los campos automáticamente del “String “que extraeremos y por cuál identificador será remplazado en “replace by R.string.”. Estos anteriores podremos modificarlos aquí, o posteriormente en el fichero “strings.xml”. Recordar, que en este misma ventana podremos indicar en qué carpeta de idioma lo guardaremos (en el cuadro “Configuration:” pasamos a la derecha “Language” donde elegiremos el idioma; si escogemos fr se nos guardará un fichero “strings.xml” con este valor en la carpeta “values-fr”. Si no ponemos nada se guardará en la carpeta por defecto “values”).



Después de esto podremos comprobar como efectivamente se nos ha cambiado el contenido del texto a un identificador de tipo String:

```
activity_main.xml
        android:paddingTop="@dimen/activity_vertical_margin'
        tools:context="com.jarroba.dibujarui.MainActivity$Pl

        <TextView
            android:id="@+id/textView$activity_main$Escribi
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="130dp"
            android:text="@string/escriba_para_guardar_"
            android:textAppearance="?android:attr/textAppear
```

Y en el fichero “strings.xml” correctamente se habrá creado:

```
strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

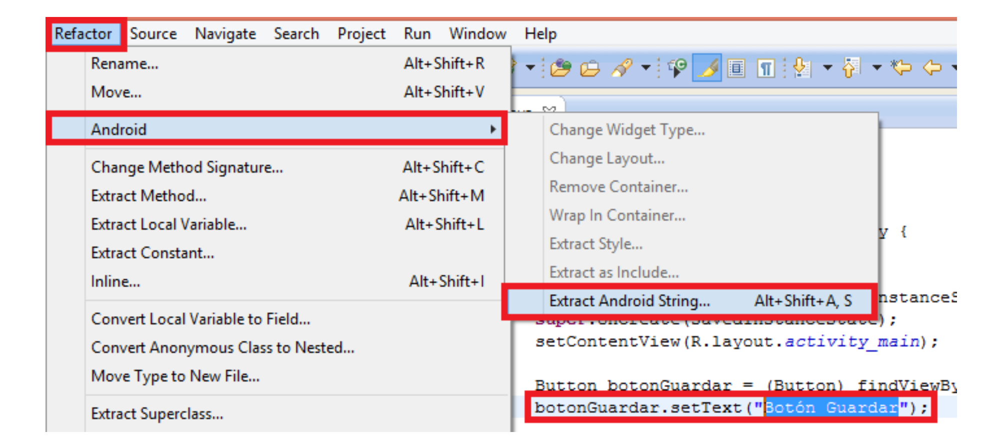
    <string name="app_name">Strings</string>

    <string name="escriba_para_guardar_">Escriba para guardar:</string>

</resources>
```

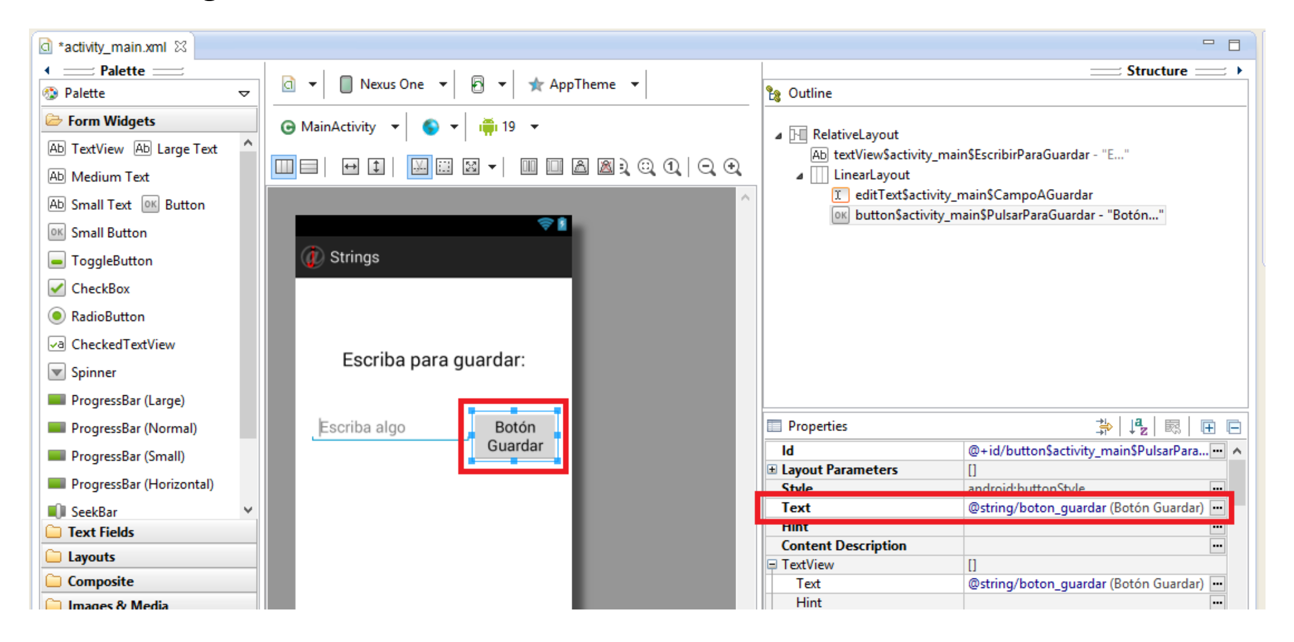
¿Y para extraerlos desde Java?

Parecido al anterior. Seleccionamos el texto y repetimos los pasos.



¿Cómo se asigna un String desde el editor gráfico de diseños XML?

Si seleccionamos la View que queremos ponerle el texto, podremos poner un String directamente sobre el atributo "Text". Si pulsamos los tres puntos, se nos abrirá un asistente para auto-crear un identificador de String para el fichero "strings.xml".



String Array

¿Qué es?

Una array de cadenas de texto

res/values/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string-array name="stringarray_pajaros">
        <item>Buho</item>
        <item>Colibrí</item>
        <item>Cuervo</item>
        <item>Flamenco</item>
        <item>Kiwi</item>
        <item>Loro</item>
        <item>Pavo</item>
        <item>Pingüino</item>
    </string-array>

</resources>
```

¿Para qué se usa?

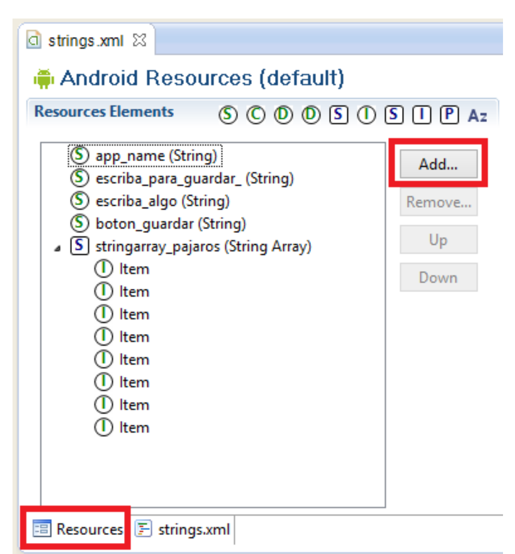
Para listados, como Spinners

¿Cómo se obtiene?

- En Java con: **getResources().getStringArray(R.array.array_strings)**

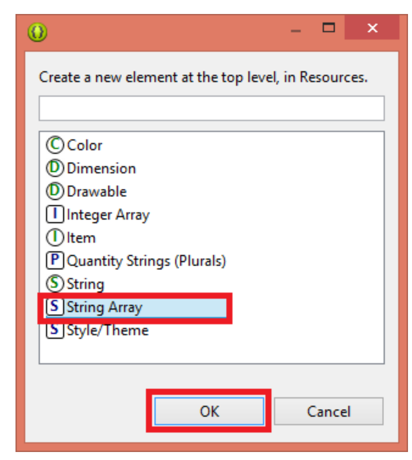
```
String[] arrayStringPajaros = getResources().getStringArray(R.array.stringarray_pajaros);
```

¿Existe una manera sencilla de añadir un Array de Strings sin tener que estar escribiendo XML?

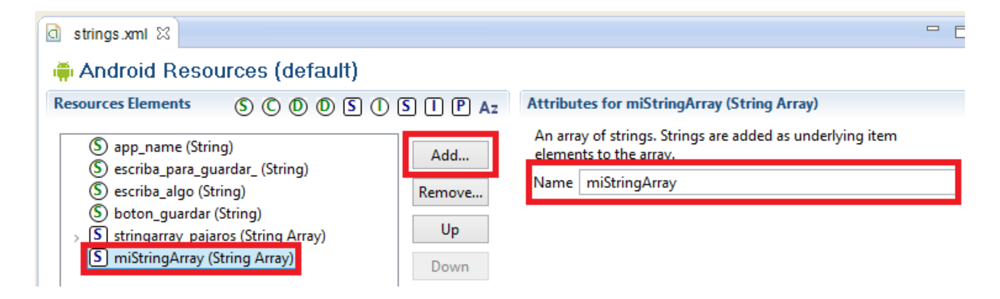


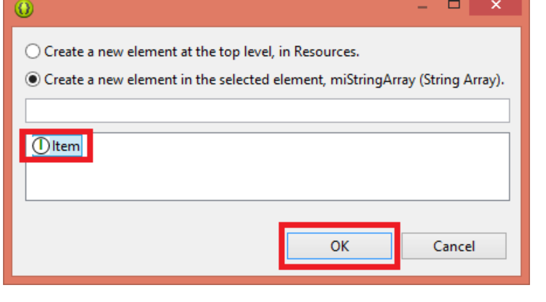
Sí, existe un asistente en el fichero “strings.xml”. Pulsamos abajo en la pestaña de “Resources”. Aquí podremos pulsar el botón “Add…”

Nos abrirá una nueva ventana en el que podremos elegir añadir “String_Array”. También permite añadir “String” simple (lo explicamos en “String Array” porque un String simple suele ser más cómodo extraerlo, aunque también vale hacerlo con este asistente o directamente sobre XML), y también nos permitirá añadir los “Quantity Strings” que se explicarán a continuación.



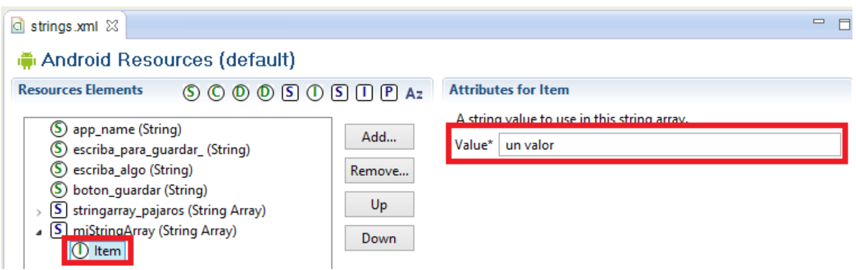
Se nos habrá creado nuestro nuevo “String Array”. Seleccionándolo le podemos poner un nombre en “Name” y luego añadir elementos pulsando otra vez “Add…”





De igual manera que antes, añadimos los elementos con “Item.

Y le ponemos un valor en “value” a cada uno.



Quantity Strings

¿Qué es?

Para poner las palabras en singular o plural, según el número

res/values/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <plurals name="plurals_numero_de_pajaros">
        <item quantity="one">Se ha seleccionado un único pájaro</item>
        <item quantity="other">Se han seleccionado %d pájaros</item>
    </plurals>

</resources>
```

¿Para qué se usa?

Para los plurales. Por ejemplo: cero libros, un libro, dos libros, tres libros, etc

¿Cómo se obtiene?

- En Java con: **getResources().getQuantityString(R.plurals.plurals_misPlurarels, contador_plural, contador_%d);**

```
for (int i = 0; i < 4; i++) {
        String stringCantidad = getResources().getQuantityString(R.plurals.plurals_numero_de_pajaros, i);
}
```

¿Qué tipos de cantidades existen?

- zero (0)
- one (1)
- two (2)
- few (pocos)
- many (muchos)
- other (otros)

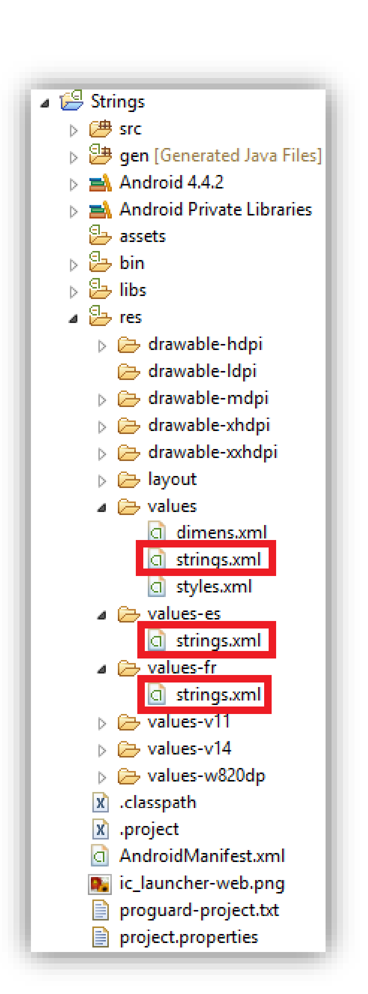
¿Cuáles se usan en español?

En español nos sirve con utilizar "**one**" para 1 y "**other**" para el resto (en español tanto cero como más de 1 se escriben en plural). El resto de tipos de cantidades son para otros idiomas.

Varios idiomas

¿Qué es?

La capacidad que tienen todas las aplicaciones Android de poder tener texto en más de un idioma.



¿Para qué se usa?

Para que personas de diferentes idiomas puedan entender, por tanto usar tu aplicación, por lo que serán potenciales clientes.

¿Cómo se obtiene?

Android permuta automáticamente entre los diferentes idiomas según esté configurado el idioma del dispositivo.

¿Cómo se añaden?

Hay que crear carpetas "values" con sufijo. Por ejemplo "values-fr" para el idioma Francés. Lo que tendrá que contener esta carpeta será una copia exacta de la estructura XML del fichero "strings.xml" que existe en la carpeta "values", con la particularidad de estar traducidos los textos a este otro dicho idioma.

¿Cómo puedo probar los idiomas en el emulador o en un dispositivo físico?



Ir al apartado de "Settings" (Ajustes) de Android.

Entrar en "Language & input en inglés" (Idioma e introducción).



Seleccionar "Language" (Idioma).

Ahí elegiremos el idioma que queramos utilizar para probar (ojo, que al cambiar el idioma, si no lo entendemos, nos tendremos que acordar de los pasos para volver a cambiarlo).

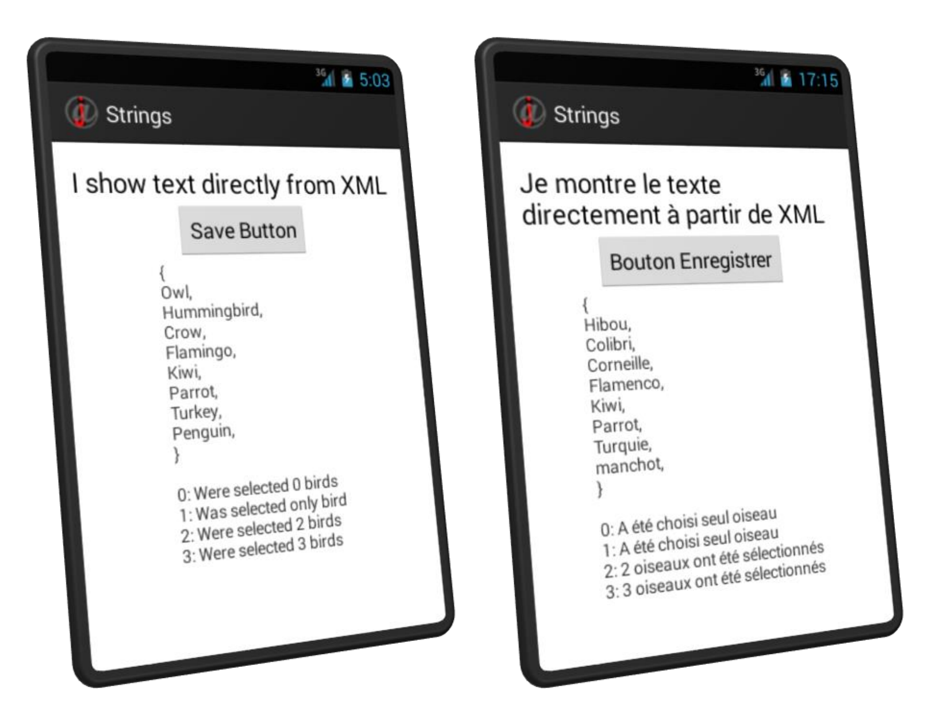
Ejemplo de Strings con idiomas

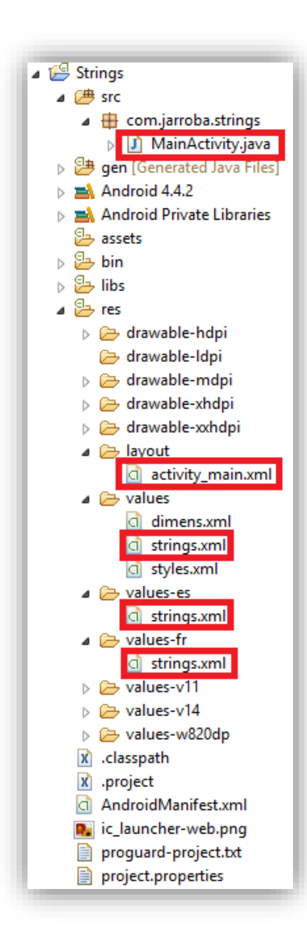
Lo que haremos

- Primero pondremos un TextView al que le pondremos el String desde el XML, asociado a un String del fichero “strings.xml”.
- Luego pondremos un botón al que se le pondremos el String desde Java, siempre tomando el String desde el fichero “strings.xml”.
- A continuación crearemos un String Array en el fichero “strings.xml”, que recorreremos y mostraremos en un TextView
- Del mismo modo, mostraremos en un TextView un String de cantidad en le que contaremos desde cero a tres



- Y para rematar, traduciremos el fichero de “strings.xml” al inglés y al francés. Y estableceremos el fichero en inglés como el idioma por defecto (lo guardaremos en la carpeta “values”).





MainActivity.java

```
public class MainActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		//-------------------String Normal----------------------
		Button botonGuardar = (Button) findViewById(R.id.button$activity_main$PulsarParaGuardar);
		String textoDelBoton = getString(R.string.boton_guardar);
		botonGuardar.setText(textoDelBoton);
		//-------------------String Normal----------------------

		String misStrings = "{";

		//-------------------String Array-----------------------
		String[] arrayStringPajaros = getResources().getStringArray(R.array.stringarray_pajaros);
		for (String pajaro : arrayStringPajaros) {
			misStrings += "\n" + pajaro + ",";
		}
		//-------------------String Array-----------------------

		misStrings += "\n}\n";

		//-------------------Quantity Strings-------------------
		for (int i = 0; i < 4; i++) {
			String stringCantidad =
getResources().getQuantityString(R.plurals.plurals_numero_de_pajaros, i, i);
			misStrings += "\n" + i + ": " + stringCantidad;
		}
		//-------------------Quantity Strings-------------------

		TextView tvMostrarStrings = (TextView)
findViewById(R.id.textView$activity_main$MostarStrings);
		tvMostrarStrings.setText(misStrings);
	}

}
```

res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.jarroba.dibujarui.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView$activity_main$EscribirParaGuardar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="@string/muestro_un_texto_directamente_desde_xml"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$activity_main$PulsarParaGuardar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView$activity_main$EscribirParaGuardar"
        android:layout_centerHorizontal="true" />

    <TextView
        android:id="@+id/textView$activity_main$MostarStrings"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button$activity_main$PulsarParaGuardar"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Strings</string>

    <string name="muestro_un_texto_directamente_desde_xml">I show text directly from
XML</string>

    <string name="boton_guardar">Save Button</string>

    <string-array name="stringarray_pajaros">
        <item>Owl</item>
        <item>Hummingbird</item>
        <item>Crow</item>
        <item>Flamingo</item>
        <item>Kiwi</item>
        <item>Parrot</item>
        <item>Turkey</item>
        <item>Penguin</item>
    </string-array>

    <plurals name="plurals_numero_de_pajaros">
        <item quantity="one">Was selected only bird</item>
        <item quantity="other">Were selected %d birds</item>
    </plurals>

</resources>
```

res/values/strings-es.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Strings</string>

    <string name="muestro_un_texto_directamente_desde_xml">Muestro un texto directamente desde
XML</string>

    <string name="boton_guardar">Botón Guardar</string>

    <string-array name="stringarray_pajaros">
        <item>Buho</item>
        <item>Colibrí</item>
        <item>Cuervo</item>
        <item>Flamenco</item>
        <item>Kiwi</item>
        <item>Loro</item>
        <item>Pavo</item>
        <item>Pingüino</item>
    </string-array>

    <plurals name="plurals_numero_de_pajaros">
        <item quantity="one">Se ha seleccionado un único pájaro</item>
        <item quantity="other">Se han seleccionado %d pájaros</item>
    </plurals>

</resources>
```

res/values/strings-fr.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Strings</string>

    <string name="muestro_un_texto_directamente_desde_xml">Je montre le texte directement à
partir de XML</string>

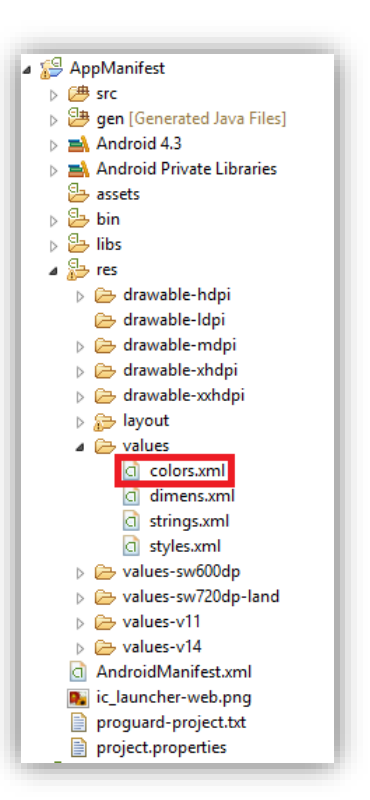
    <string name="boton_guardar">Bouton Enregistrer</string>

    <string-array name="stringarray_pajaros">
        <item>Hibou</item>
        <item>Colibri</item>
        <item>Corneille</item>
        <item>Flamenco</item>
        <item>Kiwi</item>
        <item>Parrot</item>
        <item>Turquie</item>
        <item>manchot</item>
    </string-array>

    <plurals name="plurals_numero_de_pajaros">
        <item quantity="one">A été choisi seul oiseau</item>
        <item quantity="other">%d oiseaux ont été sélectionnés</item>
    </plurals>

</resources>
```

Colors (Colores)



¿Qué es?

Es un fichero XML que llamaremos "colors.xml". Guardará los colores que definiremos en alguno de los formatos de color soportados para que los podamos utilizar sin tener que recordar la combinación hexadecimal, y para hacer independizar los colores del código.

¿Qué formatos de color soporta?

Los formatos de colores soportados son (Siendo: A alfa, R rojo, V verde y B azul):

- #RGB
- #ARGB
- #RRGGBB
- #AARRGGBB

Como recomendación y por estandarizar en el que más juego nos ofrece, es preferible utilizar siempre #AARRGGBB. Por ejemplo, para el color azul sin que transparente escribiremos: #FF0000FF

¿Cómo se crea un fichero "colors.xml"?

Sencillo, tan solo hay que escribir dentro de las etiquetas <resources> los colores que queramos con <color>. Tendrá un atributo "name" donde pondremos el nombre del color, y en el valor de la etiqueta pondremos el código hexadecimal antes explicado.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="Rojo">#FFFF0000</color>
    <color name="Verde">#FF00FF00</color>
    <color name="Azul">#FF0000FF</color>
    <color name="Amarillo">#FFFFFF00</color>
    <color name="Morado_ligeramente_transparente">#AAC54B8C</color>
    <color name="Dorado">#FFFFD700</color>
    <color name="Rosa">#FFFF00FF</color>
</resources>
```

¿Cómo se obtiene?

Se puede usar en cualquier atributo de cualquier View que permita un color, como cambiar el color de un texto.

- En los recursos XML con: **@color/color_elegido**

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textColor="@color/mi_color_azul" />
```

- En Java con: **getResources().getColor(R.color.color_elegido)**

```
int colorElegido = getResources().getColor(R.color.mi_color_azul);

TextView tv = (TextView) findViewById(R.id.TextView_unTexto);
tv.setTextColor(colorElegido);
```

¿Puedo agrupar colores?

Sí, puedes utilizar un “integer-array” para agrupar colores. Este “integer-array” también puedes utilizarlo para agrupar otros números enteros que sean finales.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="Rojo">#FFFF0000</color>
    <color name="Verde">#FF00FF00</color>
    <color name="Amarillo">#FFFFFF00</color>
    <color name="Azul">#FF0000FF</color>

    <integer-array name="ColoresSemaforo">
        <item>@color/Rojo</item>
        <item>@color/Amarillo</item>
        <item>@color/Verde</item>
    </integer-array>

</resources>
```

Para llamarlos desde Java es tan sencillo como obtener el array y utilizarlo de manera normal:

```
int[] coloresSemaforo = getResources().getIntArray(R.array.ColoresSemaforo);

int colorElegido = coloresSemaforo[2]; //Muestra el color verde
```

¿Hay algún color que pueda usar, para no tener que crearlos cada vez?

Se pueden utilizar los colores que trae la biblioteca de Android, así nos ahorramos líneas de código.

- En los recursos XML con: **@android:color/color_elegido**

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textColor="@android:color/holo_purple" />
```

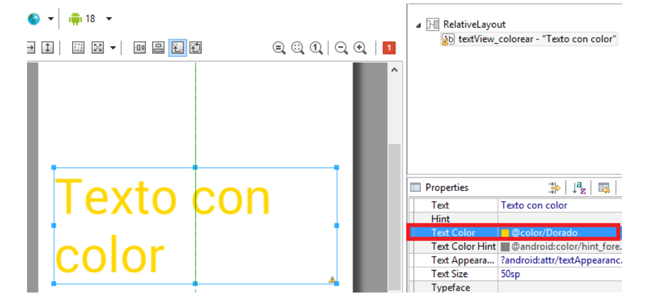
- En Java con: **getResources().getColor(android.R.color.color_elegido)**

```
int colorElegido = getResources().getColor(android.R.color.holo_purple);
```

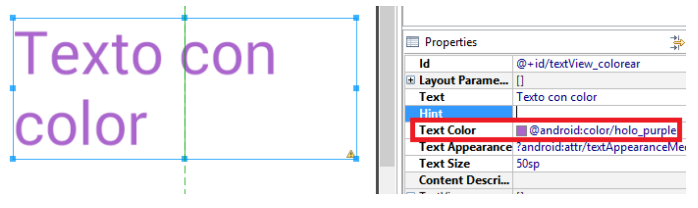
¿Se puede asignar desde el editor gráfico de diseños?

Sí, desde el panel de “Properties” del editor gráfico podremos elegir un atributo que admita colores, buscarlo y seleccionarlo.

Ejemplo de color definido por nosotros en “**colors.xml**”:



Ejemplo de color utilizando los que vienen con la **biblioteca de Android**:



Referencias:

- http://developer.android.com/guide/topics/resources/more-resources.html

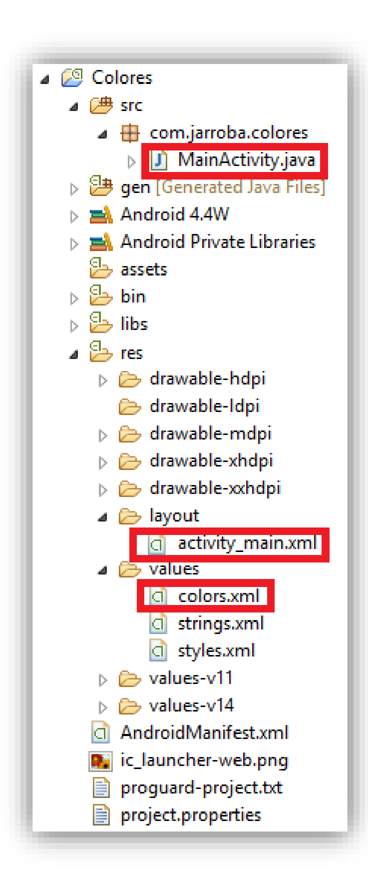
Ejemplo de uso de colores

Lo que haremos

- Un texto coloreado con uno de nuestros colores desde el diseño XML.
- Un texto coloreado con un color del API de Android desde el diseño XML.
- Un texto coloreado con uno de nuestros colores desde el diseño Java.
- Un texto de un botón coloreado con un color del API de Android desde el diseño Java.
- Un texto que tenga cada letra coloreada con tres colores diferentes, utilizando un “integer-array”. Este texto lo colorearemos gracias a la clase “Spannable” -tenemos que pensar en esta clase como si de un “String” se tratara- la gracia de este String de tipo Spannable es que le podemos aplicar formatos un formato diferente a cada letra del String (Spannable signifique en español “abarcable”, pues con el método setSpan() iremos abarcando grupos de caracteres para darles estilo; en este ejemplo iremos abarcando de carácter en carácter). Una pista es hacer un “for” que recorra cada letra y le aplique un color diferente a cada una. En este ejemplo utilizaremos el módulo “%” de Java para facilitarnos el recorrer el array; como recordatorio de este operador “%” es lo que da el resto de la división (ejemplo: 0%3=**0**, 1%3=**1**, 2%3=**2**, 3%3=**0**, 4%3=**1**, 5%3=**2**, 6%3=**0**…).
- Un fondo coloreado con uno de nuestros colores desde el diseño XML. Utilizaremos el “LinearLayout” como fondo en este ejemplo, aunque luego veremos cómo se hace con estilos.

Nota: por facilitar la comprensión del código he puesto los Strings a pelo en el código, recuerda utilizar siempre los ficheros de “strings.xml”





MainActivity.java

```
public class MainActivity extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                // Coloreamos el texto con un color de los nuestros
                int colorMio = getResources().getColor(R.color.Morado_ligeramente_transparente);

                TextView tv = (TextView) findViewById(R.id.textView$activity_main$texto);
                tv.setTextColor(colorMio);

                // Coloreamos el texto con un color de la API de Android
                int colorDeLaApiDeAndroid = getResources().getColor(android.R.color.holo_blue_dark);

                Button btn = (Button) findViewById(R.id.button$activity_main$boton);
                btn.setTextColor(colorDeLaApiDeAndroid);

                // Colorearemos cada letra de un texto con el array
                int[] coloresSemaforo = getResources().getIntArray(R.array.ColoresSemaforo);

                Spannable textoColoreado = new SpannableString("Texto que tendrá cada letra coloreada");
                for (int i = 0; i < textoColoreado.length(); i++) {
                        Log.v("test", i + " % 3 = " + (i % 3));
                        int colorElegido = coloresSemaforo[i % 3];
                        textoColoreado.setSpan(new ForegroundColorSpan(colorElegido), i, i + 1,
Spannable.SPAN_INCLUSIVE_INCLUSIVE);
                }

                TextView tvColores = (TextView)
findViewById(R.id.textView$activity_main$textoLetrasColoreadas);
                tvColores.setText(textoColoreado);
        }

}
```

res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/DoradoClaro"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.colores.MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Coloreado desde XML con nuestro color"
        android:textColor="@color/Rojo"
        android:textSize="20sp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Coloreado desde XML con un color de la API de Android"
        android:textColor="@android:color/holo_green_dark"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/textView$activity_main$texto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Texto coloreado desde Java con nuestro color"
        android:textSize="20sp" />

    <Button
        android:id="@+id/button$activity_main$boton"
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:text="Texto de botón coloreado desde Java con un color de la API de Android"
        android:textSize="15sp" />

    <TextView
        android:id="@+id/textView$activity_main$textoLetrasColoreadas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Texto coloreado desde Java con nuestro color"
        android:textSize="20sp" />

</LinearLayout>
```

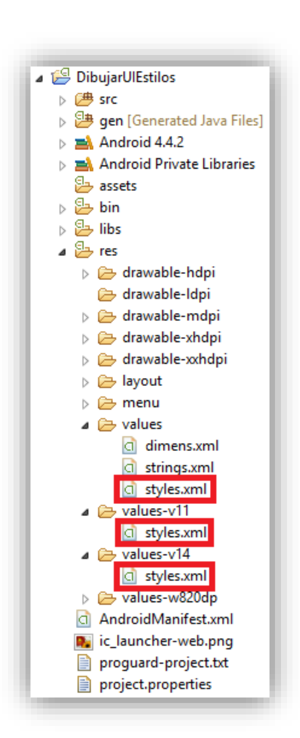
res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="Rojo">#FFFF0000</color>
    <color name="Verde">#FF00FF00</color>
    <color name="Azul">#FF0000FF</color>
    <color name="Amarillo">#FFFFFF00</color>
    <color name="Morado_ligeramente_transparente">#AAC54B8C</color>
    <color name="DoradoClaro">#FFF6D574</color>

    <integer-array name="ColoresSemaforo">
        <item>@color/Rojo</item>
        <item>@color/Amarillo</item>
        <item>@color/Verde</item>
    </integer-array>

</resources>
```

Styles (Estilos)



¿Qué es?

Un estilo es una colección de propiedades que especifican aspecto y el formato de una View, Activity, Fragment o a toda la aplicación. Dicho de otro modo, sirve para no tener que repetir los mismos atributos en todas las Views. Un ejemplo sería si quisiéramos que todos los TextView de nuestra aplicación el texto fuera de color verde, con un tamaño de letra de 20sp y en negrita; podríamos ir TextView en TextView copiando y pegando los mismos atributos.

¿Cuáles hay?

Hay bastantes. Algunos son: color, font (fuente), padding (relleno), size (tamaño), height (altura), etc.

¿Se declara en otro fichero aparte del de diseño?

Sí, se definirá en la carpeta “res” dentro de alguna de las carpetas “values” (dentro de la carpeta “values” dependiendo de características deba tener el dispositivo a aplicar el estilo; para empezar, recomiendo trabajar solo con la carpeta llamada tal cual “values”, la que no tiene extensión alguna). El fichero de estilos se puede llamar como queramos y con la extensión “.xml”; al crear un proyecto ya existe uno llamado “styles.xml” que podemos utilizar.

¿Es útil separar los estilos de los layout?

Mucho. Una ventaja es que podremos tener varios estilos para cada tipo de dispositivo (por la versión del sistema operativo, por el tamaño de la pantalla, si es Tablet o Smartphone, etc). Otra es la independencia que conseguimos y la facilidad que nos aportará para realizar cambios en un futuro al disponer del contenido en la carpeta “layout” y los estilos en las carpetas “values”.

Supongamos que queremos poner dos TextViews de color rojo, con un tamaño de 20sp, con 10dp de margen, ajustados al contenido tanto por alto como pon ancho.



Si aplicáramos los estilos directamente sobre las Views tendríamos un diseño en la carpeta “layout” como el que se muestra a la derecha.

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="TextView A - www.Jarroba.com"
        android:textColor="#FFFF0000"
        android:textSize="20sp"
        />

<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="TextoView B - Tutoriales Informáticos"
        android:textColor="#FFFF0000"
        android:textSize="20sp" />
```

Podemos ver que es mucho código, y si algún día tuviéramos que cambiar todos los textos, a por ejemplo, a color azul, habría que ir de uno en uno.

Por otro lado, si aplicáramos estilos tendríamos el diseño de la carpeta layout que está al lado de este párrafo. Mucho más pequeño el XML y al cambiar algo del fichero estilos cambiará en todos a la vez.

```
<TextView
        style="@style/MiEstiloDeLosTextViews"
        android:text="@string/jarroba" />

<TextView
        style="@style/MiEstiloDeLosTextViews"
        android:text="@string/jarroba_b" />
```

Ver siguiente pregunta el ejemplo del estilo que se aplica.

¿Cómo se escribe un fichero de estilos en XML?

Definiremos el fichero de estilos en formato XML en la carpeta “values” apropiada para que se apliquen dependiendo de si nos interesa a todo o solo en Tablets, Smartphones, dependiendo de la versión del sistema operativo, entre otras.

Cada fichero de estilos se formará con estas etiquetas:

- **<resources>**: Los estilos son recursos, por lo que se escriben entre las etiquetas <resources>. Podrá tener varios <style>
- **<style>**: Cada estilo estará contenido en una etiqueta <style>. Este tendrá un name (nombre) que queramos, que será el referido por la View con “@style/nombre_estilo”. Y opcionalmente -aunque muy recomendado- un parent (padre) para poder reutilizar los estilos de éste y luego modificar los que queramos. Podrá tener varios <ítem>
- **<item>**: Cada modificación del estilo estará entre las etiquetas <item>. Tendrán un name (nombre) que ha de coincidir con la propiedad que queramos modificar, siempre comenzando con “android:”. Entre la etiqueta de apertura y cierre pondremos el valor de la propiedad que utilizar la View que utilice estos estilos.

Completando al ejemplo anterior, para dar estilos a las Views que solicitan estilos externos. Este es el código de los estilos que utiliza, que está en la carpeta “values”.

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">

    <style name="MiEstiloDeLosTextViews" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:layout_margin">10dp</item>
        <item name="android:textColor">#FFFF0000</item>
        <item name="android:textSize">20sp</item>
    </style>

</resources>
```

¿Son parecidos los estilos en Android a las CSS en diseño web?

La filosofía de separar el diseño del contenido es la misma. Además que se aplican los estilos en cascada; es decir, se aplica el último más próximo al elemento (un ejemplo sería que definimos una hoja de estilos para que todos los textos sean de color rojo, y hay justo un texto que en exclusiva queremos que sea azul, si definimos el azul en este texto, se aplicará el azul y no el rojo que está en la hoja de estilos).

¿Es lo mismo un estilo que un tema?

Un tema es un estilo. Tema tiene un significado adicional, que indica que será aplicado a toda la Activity o a una aplicación en el AndroidManifest.xml, en vez de ir aplicando un estilo de View en View.

¿Puedo aplicar el mismo estilo a toda la aplicación o a toda una Activity?

Claro que sí. Habrá que añadirlo al AndroidManifest.xml a la etiqueta <application> para que se aplique a toda la aplicación, o a la etiqueta <Activity> para que afecte solo a una Activity en concreto.

Por ejemplo, la aplicación ya está utilizando por el tema llamado “AppTheme” que está definido en el fichero “styles.xml” dentro de la carpeta “values”. Podremos modificarlo desde el fichero “styles.xml” y aprovechar que ya está heredando del tema de la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.dibujaruiestilos"
    android:versionCode="1"
    android:versionName="1.0" >

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
    </application>

</manifest>
```

¿Hay que definir todos los atributos aunque algunos los quiera mantener a los de por defecto?

Como se explicó anteriormente no es necesario. Opcionalmente se puede añadir un estilo al "parent" para poder reutilizar los estilos de éste y luego modificar los que queramos.

¿Puedo heredar mis propios estilos?

Puedes heredar tus propios estilos para no tener que repetir la declaración de atributo.

Si por ejemplo, un estilo que ponía los textos en rojo se llamaba "textoEnRojo" y quieres que otro estilo sea en rojo y en negrita. Con llamar al nuevo estilo con el mismo nombre del anterior, seguido de punto, y el nombre que queramos, como "textoEnRojo.negrita", creamos un nuevo estilo heredando todos los atributos de nuestro otro estilo.

Referencias:

- http://developer.android.com/guide/topics/resources/style-resource.html
- http://developer.android.com/guide/topics/ui/themes.html

Ejemplo de aplicar estilos

Lo que haremos

Vamos a tener varios TextViews que han de tener:

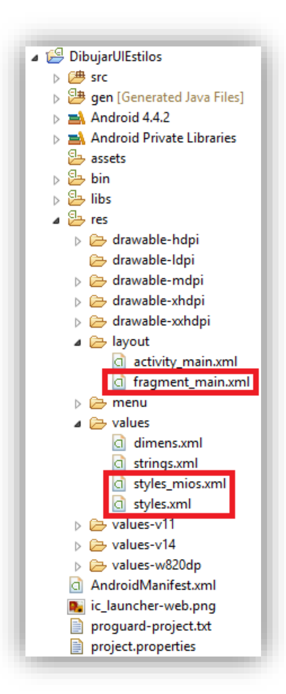
- color rojo
- un tamaño de 20sp
- 10dp de margen
- ajustados al contenido tanto por alto como pon ancho

Algunos TextViews de ellos, aparte de todo lo anterior, además van a ser en negrita.

También aplicaremos un fondo verde a toda la aplicación.



Ilustración 13 - Ejemplo con dos TextView, uno con los estilos pedidos y otro además en negrita



res/layout/fragment_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

     <TextView
         style="@style/MiEstiloDeLosTextViews"
         android:text="@string/jarroba" />

    <TextView
        style="@style/MiEstiloDeLosTextViews.negrita"
        android:text="@string/jarroba_b" />

</LinearLayout>
```

res/values/styles_mios.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">

     <style name="MiEstiloDeLosTextViews" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:layout_margin">10dp</item>
        <item name="android:textColor">#FFFF0000</item>
        <item name="android:textSize">20sp</item>
    </style>

    <style name="MiEstiloDeLosTextViews.negrita">
        <item name="android:textStyle">bold</item>
    </style>

</resources>
```

res/values/styles.xml

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">

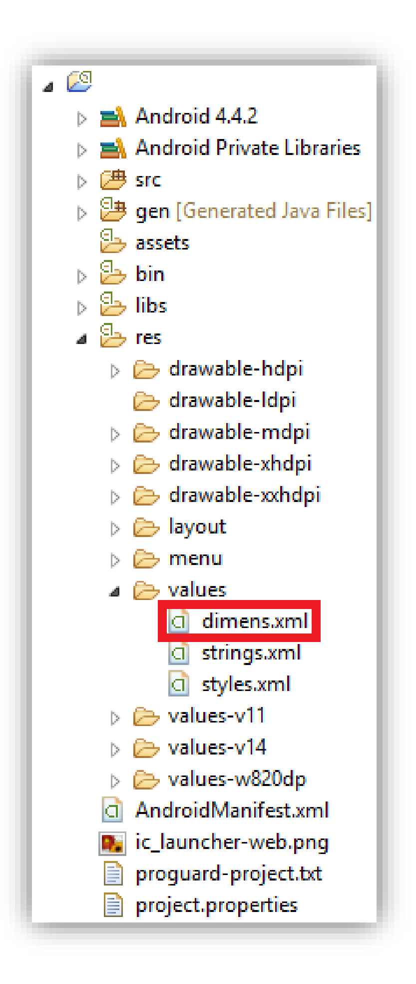
    <style name="AppBaseTheme" parent="android:Theme.Light">
    </style>

    <style name="AppTheme" parent="AppBaseTheme">
        <item name="android:windowBackground">@android:color/holo_green_light</item>
    </style>

</resources>
```

Ficheros para la externalización de recursos

¿Qué son?



Ya hemos visto algunos como “strings.xml” o “colors.xml”, existen otros. Son ficheros XML que sirven para extraer del código ciertos valores. Por tanto, independizamos el código de los posibles valores que pueda tomar. Además, de tener varias veces el mismo valor repartido por diferentes ficheros o partes del código; al cambiarlo únicamente en estos ficheros, conseguiremos fácilmente cambiar el valor en todos los sitios de manera apropiada, y con un coste ínfimo de tiempo.

¿Cuáles existen?

- Bool: valores de tipo boolean
- Color: valores hexadecimales que representan colores
- Dimensions: valores de dimensión (normalmente dp o sp) que representarán medidas
- ID: valores identificadores de recursos
- Integer: constantes enteras como máximos y mínimos
- Integer Array: igual que le de Integer pero en Array
- Typed Array: Array de tipos de datos mezclados

Referencias:

- http://developer.android.com/guide/topics/resources/more-resources.html

Bool (boolean)

¿Qué es?

Es un fichero que guarda constantes de tipo boolean (true o false).

¿Por qué se usa?

Para guardar un true o un false, según convenga. Un ejemplo sería el de desarrollo (por ejemplo, si lo ponemos a true mostrará los logs, si lo ponemos a false no), entre otros.

Color (colores)

¿Qué es?

Es un fichero que guarda valores hexadecimales que representan colores.

¿Por qué se usa?

Para guardar colores y no tener que memorizar, por ejemplo que #FF0000 significa rojo. Todos los valores de color tienen delante un prefijo de una almohadilla "#". Se suele utilizar el formato de color RGB (R rojo, G verde, B azul) de tipo #RRGGBB, pero también el ARGB (A alfa) poniendo #AARRGGBB

Dimens (dimensiones)

¿Qué es?

Es un fichero que guarda constantes de valores de dimensiones en XML. Sus valores se representan de un número seguido de una unidad de medida (como ya se vio anteriormente, aunque se suelen utilizar dp y sp).

¿Por qué se usa?

Para guardar medidas como pueden ser tamaño de los márgenes, tamaño de la fuente de la letra, anchura o altura de Views, etc.

Id (Identificador de recurso)

¿Qué es?

Es un fichero que guarda identificadores únicos, que se autogenerarán en el fichero "R.java".

¿Por qué se usa?

Para no tener que declarar los identificadores en los propios recursos y poder saber si estamos repitiendo alguno o no. Además, a la hora de asignarlo en un recurso hay que tener en cuenta que no lo creamos (no utilizamos el símbolo de suma, como por ejemplo "@+id/mi_id"), sino que llamamos a un identificador ya creado (sin utilizar el símbolo de suma, como por ejemplo "@id/mi_id").

Integer (Número entero)

¿Qué es?

Es un fichero que guarda una constante numérica entera con un nombre.

¿Por qué se usa?

Para guardar constantes como la velocidad máxima o mínima.

Integer Array (Colección de números enteros)

¿Qué es?

Es un fichero que guarda arrays de constantes de números enteros.

¿Por qué se usa?

Para guardar constantes, por ejemplo, un array con números primos

Typed Array (Colección de valores tipados)

¿Qué es?

Es un fichero que guarda arrays de constantes de datos de diferentes tipos.

¿Por qué se usa?

Para guardar colecciones de constantes de diferentes tipos de datos, como por ejemplo, un array con los colores que definen el arcoíris, o un conjunto de identificadores a de la carpeta “drawable” que vayan a estar en una determinada Activity.

Eventos de Java

¿Qué es la “programación dirigida por eventos” o lo que es lo mismo “programación orientada a eventos”?

Android utiliza el lenguaje Java con su compilador, por lo que soporta la “programación orientada por eventos”. Android la utiliza en todas sus facetas, por lo que necesitamos entenderla de manera muy precisa.

Este paradigma tan solo indica que no vamos a seguir una linealidad en el código, sino que se ejecutarán ciertos fragmentos de código cuando ocurra cierta acción (en un “evento” concreto).

Un ejemplo muy rápido es el de un editor de revistas y su suscriptor. El suscriptor quiere unas revistas para hacer algo con ellas, para conseguirlas, el suscriptor tiene dos opciones:

A. **Preguntar todo el rato al editor** de revistas si ya las tiene preparadas

B. **Suscribirse al editor** de revistas **y que este le avise** cuando estén listas





Si has observado la imagen entre la diferencia de la “programación secuencial” y la “programación dirigida por eventos”, seguro que te habrás dado cuenta que para ciertas acciones puede resultar muy ineficiente preguntar todo el rato. En programación sería, por ejemplo, preguntar todo el rato a un botón si ha sido pulsado para hacer algo cuando se pulse. Es decir, un bucle “while” mientras no pulsado sería muy ineficiente, además de consumir procesador y por ello batería (Además de tener que hacerlo en un hilo en segundo plano para no bloquear el programa, suena hasta complicado para hacer tan poco).

De ahí surgió la “programación dirigida por eventos”, de la necesidad de esperar a que algo ocurra sin las contras anteriores. De una manera muy sencilla podemos suscribirnos a un evento, que en el momento en el que ocurra nos avise y podamos hacer algo.

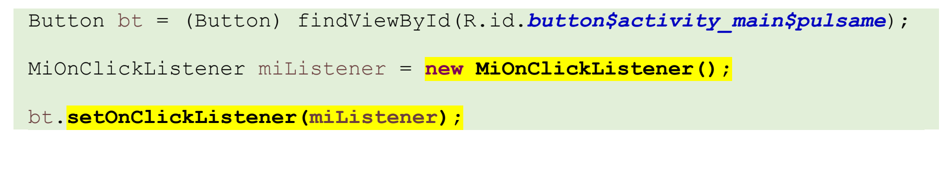
Es decir, el flujo del programa lo controlan los eventos. Eventos que pueden ocurrir tanto por acciones del usuario (por ejemplo, pulsar un botón o escribir en un campo editable), como a la espera de que termine otra parte de código (por ejemplo, esperar a que termine de descargar imágenes de internet o que finalice un hilo).

La “programación dirigida por eventos” consta de dos partes implicadas en el proceso (puedes revisar el la imagen del ejemplo anterior para comprobar que los nombres tienen sentido):

- **Editor**: emisor del evento (cuando pase algo enviará el evento)
- **Suscriptor**: consumidor del evento (al recibir el evento se encarga de controlar que va a ocurrir). Normalmente lo veremos con el nombre de “Listener” (porque un suscriptor es quien escucha a un editor a la espera de que termine el evento al que se suscribió) y con el prefijo (lo veremos más a fondo en el tema de multitarea):
    - **On**: para indicar que se ejecutará en primer plano (hilo principal encargado de controlar las vistas)
    - **Do**: para indicar que se ejecutará en segundo plano (en un hilo secundario)

¿Cómo se utilizan los eventos en Android?

Si queremos suscribir los métodos de una clase llamada “MiOnClickListener.java” al evento OnClick (evento que se lanza al pulsar el botón) del botón que será el editor, tendremos que llamar al método de suscripción apropiado. En este caso es tan fácil como pasarle nuestro objeto de tipo “MiOnClickListener.java” a suscribir a setOnClickListener().



```
Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);

MiOnClickListener miListener = new MiOnClickListener();

bt.setOnClickListener(miListener);
```



Al pasar el objeto al método “setOnClickListener()” nos fijaremos que a su vez este objeto tendrá que ser una implementación de “OnClickListener” (nos lo pedirá Eclipse). La razón es que necesitamos una interfaz común entre el editor y el suscriptor (es decir, el desarrollador contratado por Google que hizo la biblioteca de Android “OnClickListener” tiene que llamar a un método que nosotros sepamos cual es para poder utilizarlo; por esto utilizamos una implementación de una interfaz común de comunicación entre su código y el nuestro). Al pulsar el botón, el botón sabe que tiene que llamar a un método de un objeto que implemente “OnClickListener”, que en este caso es un único método “onClick()”. Sobrescribiremos el método “onClick()” cuyo interior tendrá lo que queremos que suceda al pulsarse (este código de este método “onClick()” quedará a la espera de que el usuario pulse el botón). Por lo que nuestra clase que hará algo cuando se pulse quedará (en el momento en que se pulse el botón ejecutará el código que haya en lugar del comentario):

MiOnClickListener.java

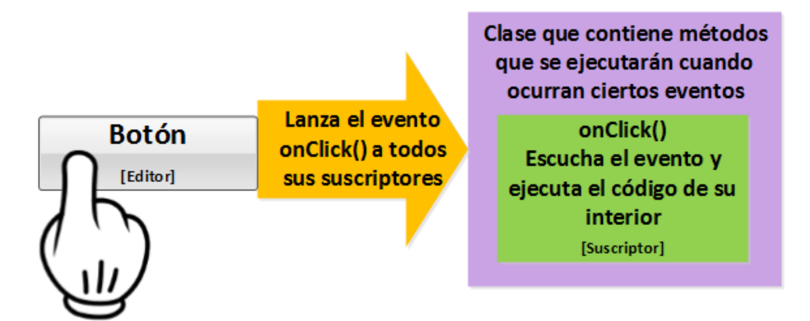
```
import android.view.View;
import android.view.View.OnClickListener;

public class MiOnClickListener implements OnClickListener {

        @Override
        public void onClick(View v) {
                //Un código a ejecutar cuando ocurra algo
        }

}
```

Al pulsar el botón, el editor (el objeto de tipo "Button") llamará al onClick() de la interfaz común que implementa de "OnClickListener". Como nuestro objeto "miListener" está escuchando, será llamado su método onClick() y por tanto su código interno.



¿Qué posibilidades nos ofrece Android para escuchar un evento de una View? ¿Existe alguna manera que no suponga crear más ficheros de clases para programar eventos?

Respondo a las dos preguntas con las siguientes maneras (nota: a la hora de importar se darán a elegir entre varias bibliotecas "OnClickListener", para estos ejemplos utilizamos "android.view.View.OnClickListener"):

- **Clase que implementa una interfaz** de un Listener: es igual que hicimos en el ejemplo anterior, solo que podemos utilizar implementar directamente de una Activity, un Fragment, o lo que queramos; para evitar tener clases que únicamente se utilizan en una de nuestras clases. Luego para suscribir el evento al setOnClickListener() bastará con pasar la misma clase con un "this". Un ejemplo sería:

MainActivity.java

```
public class MainActivity extends Activity implements OnClickListener {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);

                bt.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
                // Un código a ejecutar cuando ocurra algo
        }

}
```

- **Clase anónima**: Los anteriores ejemplos de implementar una clase están muy bien y funciona perfectamente. Pero no es la manera más corta y de mejor lectura (en la anterior pregunta lo he explicado para que se entienda el conjunto de lo que pasaré a resumir). Podemos hacer lo mismo ahorrando mucho código y de una manera más comprensible (así será como lo verás los códigos por Internet). Sabemos que la clase "MiOnClickListener.java" solo servirá para ser suscrita a un único editor, por lo que podemos utilizar una clase anónima (clase sin nombre que crea un objeto en un único instante del código y que no se reutilizará porque no hace falta). Con el siguiente poco de código ya estamos escuchando a nuestro botón:

```
Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);

bt.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                // Un código a ejecutar cuando ocurra algo
        }
});
```

- **Desde una View** en el editor: podemos asignar directamente desde XML un método con el nombre que queramos que se ejecute cuando el botón sea pulsado. Las pegas de este modo es que no todos los eventos se pueden asignar así; y que el autocompletar no funciona, por lo que no nos podrá ayudar a la hora de completar el código. Para el diseño XML tenemos que asignar ponerle al botón la propiedad "android:onClick" con el nombre del método que tendrá el suscriptor.

res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >

    <Button
        android:id="@+id/button$activity_main$pulsame"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="70dp"
        android:onClick="onPulsame"
        android:text="Púlsame" />

</RelativeLayout>
```

Y para el código de Java nos podemos ahorrar buscar al botón, ya que está suscrito desde el XML. Necesitamos que en la clase exista un método, exactamente con el mismo nombre que pusimos en la propiedad "android:onClick", y que le llegue una View como parámetro.

MainActivity.java

```
public class MainActivity extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
        }

        public void onPulsame(View v) {
                // Un código a ejecutar cuando ocurra algo
        }

}
```

Como guía, si estás aprendiendo como se manejan los eventos te recomiendo que utilices el primer ejemplo de la anterior pregunta (crear una clase aparte con la implementación) o "Clase que implementa una interfaz de un Listener" para entender bien el funcionamiento de los eventos. A los pocos usos, que te veas que manejas los eventos con soltura, podrás pasar a la "clase anónima" que personalmente recomiendo.

¿Cómo afecta a la utilización de variables dentro de clases anónimas?

La teoría de Java nos dice que una clase anónima:

1. Tiene acceso a los miembros de la clase que la encierra.
2. No tiene acceso a variables locales en el alcance que lo encierra (enclosing scope), a menos que estén declaradas como "final" (variable imposible de cambiar una vez declarada, inmutable). Es decir, que si no son "final", la clase anónima no podrá acceder a las variables del método que la contiene.

3. Las variables declaradas dentro serán sombreadas (shadowing). Es decir, si dentro de la clase anónima declaramos una variable con el mismo nombre, que una ya existente que en la clase que encierra, se hará caso a la declarada dentro de la clase anónima y no a la de fuera.

Esta teoría tiene su lógica cuando hablamos de eventos. Una cosa inherente a los eventos e que tendremos código esperando por su ejecución y código que ya ha sido ejecutado. En los ejemplos anteriores tiene que quedar claro que existirá código ya ejecutado en el pasado, cuando en el futuro el usuario quiera pulsar botón (quién sabe si tardará milisegundos u horas, la cosa es que hay código que ya ha sido ejecutado y otro que está esperando).

El siguiente código incumple punto 2 de la teoría de Java sobre clases anónimas, arrojando un error en tiempo de compilación (error que dirá algo así como: "Cannot refer to the non-final local variable unTexto defined in an enclosing scope"):

```
String unTexto = "escrito fuera del evento";

Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);
bt.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                unTexto = "escrito dentro del evento";
        }
});
```

Por lo que hemos dicho antes, la variable está declarada cuando la ejecución ya pasó y solo se puede utilizar en el código secuencial onCreate().

La solución del punto 2 de la teoría es hacer final a la variable para que no se pueda modificar fuera del evento:

```
final String unTexto = "escrito fuera del evento";

Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);
bt.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                unTexto = "escrito dentro del evento";
        }
});
```

Pero lo anterior puede no satisfacernos, ya que no podremos modificar la variable y puede que sí queramos modificarla. Otra aproximación es la de escribir las variables como globales, para poder utilizarlas dentro y fuera del evento (Como dice el punto 1 de la teoría). Podremos modificarla fuera del evento, y al momento en que alguien pulse el botón dentro del evento.

MainActivity.java

```
public class MainActivity extends Activity {

        private String unTexto;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                unTexto = "escrito fuera del evento";

                Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);
                bt.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                unTexto = "escrito dentro del evento";
                        }
                });

        }

}
```

Ahora bien, para entender el tema del sombreado (punto 3 de la teor) necesitamos ver en cuántos sitios podemos declarar variables y de este modo poder utilizarlas en nuestro, que son:

- Variable global: podemos evitar el sombreado llamándola con "Clase.this.variableGlobal"

- Variable local de método: no se puede evitar el sombreado
- Variable global de clase anónima: podeos evitar el sombreado llamándola con "this.variableGlobalAnonima" (dentro de una clase anónima, el "this" apunta a todo el alcance de la clase anónima, no fuera de ésta)
- Variable local de método de clase anónima: es la que sombrea, por lo que la podemos llamar directamente

Un ejemplo de cada una de estas sería:

MainActivity.java

```
public class MainActivity extends Activity {

	private String unTexto = "variable global de la clase externa";

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		final String unTexto = "variable local del método de la case externa (será sombreada, por tanto no se utilizará)";

		Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);
		bt.setOnClickListener(new OnClickListener() {

			private final String unTexto = "variable global de la clase anónima";

			@Override
			public void onClick(View v) {
				String unTexto = "variable local del método de la clase anónima";

				Log.v("test", unTexto);
				Log.v("test", this.unTexto);
				Log.v("test", MainActivity.this.unTexto);
			}
		});

	}
}
```

Y cuando pulsáramos el botón nos devolvería el logcat la siguiente salida:

```
-variable local del método de la clase anónima
-variable global de la clase anónima
-variable global de la clase externa
```

¿Por qué el método "onClick" le llega como parámetro un objeto de tipo View?

Esta View Android nos la pasa como parámetro de "onClick(**View v**)" para facilitarnos todavía más la vida a los desarrolladores (para evitar tener que declararla como global o final, por la teoría de las clases anónimas de la anterior pregunta). Esta View devuelta es en realidad la View editora del evento. En el caso de los anteriores ejemplos es el mismo Button el que nos devuelve la View. Como recordatorio sabemos que Button extiende de View, por lo que tendremos hacer un cast de la View a Button para poder utilizarla y luego ya podremos hace lo que queramos con la variable de tipo Button dentro del evento. Ejemplo de cambiar el texto al botón:

```
Button bt = (Button) findViewById(R.id.button$activity_main$pulsame);
bt.setOnClickListener(new OnClickListener() {

	@Override
	public void onClick(View v) {
		Button btDentro = (Button) v;
		btDentro.setText("Botón pulsado");
	}

});
```

¿Todo esto solo funciona para "onClick"?

Hemos explicado solo un Listener de los muchos que hay. Todos funcionan igual a la hora de programarlos en Java, aunque se ejecutarán para eventos diferentes.

¿Se pueden usar expresiones Lamda de Java para los eventos?

Las expresiones Lamda se incorporaron en el JDK 1.8 de Java. Android de momento solo compila hasta el JDK 1.7, por lo que de momento no. Presumiblemente se podrá en un futuro.

Referencias:

- http://es.wikipedia.org/wiki/Arquitectura_dirigida_por_eventos
- http://es.wikipedia.org/wiki/Programaci%C3%B3n_dirigida_por_eventos
- http://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html

Ejemplo simple de utilización de eventos

Lo que haremos

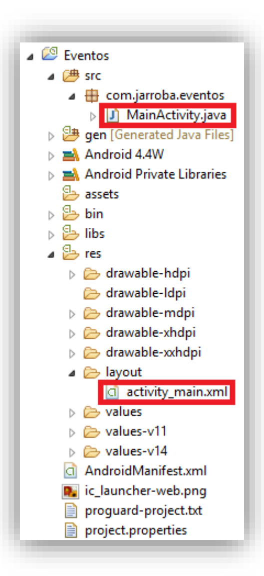
Al pulsar el botón cambiaremos el color a un texto y cambiaremos su texto. Además, el botón también cambiará de texto y se quedará deshabilitado para que no se pueda volver a pulsar.



Ilustración 14 - Estado inicial del programa



Ilustración 15 - Después de pulsar el botón



MainActivity.java

```java
public class MainActivity extends Activity {

	private Button bt;
	private TextView tv;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		tv = (TextView) findViewById(R.id.textView$activity_main$coloreame);

		bt = (Button) findViewById(R.id.button$activity_main$pulsame);
		bt.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				Button btDentro = (Button) v;
				btDentro.setText("Pulsado y color cambiado");
				btDentro.setEnabled(false);

				int color = getResources().getColor(android.R.color.holo_red_light);
				tv.setTextColor(color);
				tv.setText("Color cambiado");
			}
		});
	}
}
```

res/drawable-mdpi/estados_del_boton.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >

    <Button
        android:id="@+id/button$activity_main$pulsame"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="130dp"
        android:text="Púlsame y cambia el color al texto" />

    <TextView
        android:id="@+id/textView$activity_main$coloreame"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button$activity_main$pulsame"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="36dp"
        android:text="Cámbiame de color"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

Pasar eventos desde Fragments

¿Para qué quiero pasar eventos desde un Fragment?

Supón que tienes un listado en un Fragment a la izquierda, y a la derecha tienes el detalle del listado en otro Fragment ¿Cómo pasarías la información de que el usuario a pulsado sobre una fila del Fragment del listado al Fragment del detalle? Lo veremos aquí y ampliaremos en temas más adelante.

¿Cómo se pasan eventos de un Fragment a fuera de esta?

Mediante eventos. Te respondo con la propia pregunta, ya que necesitaremos crear nuestro propio editor que será el Fragment que quiere comunicar el evento en cierto momento. Por lo que necesitaremos gestionar nuestros propios callbacks.

¿Se puede pasar también información?

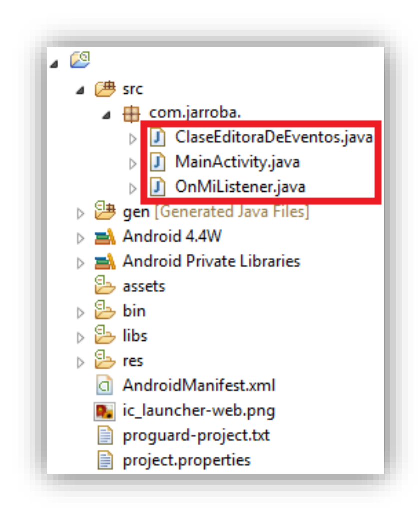
Claro que sí, al fin y al cabo un evento es la información sobre un cambio de estado. Como tenemos que crear nuestros métodos le podremos pasar lo que queramos.

¿Cómo implementar nuestro propio editor (callback o listener)?

Se hace con interfaces. Necesitaremos lo siguiente:

- Una **interfaz**: la interfaz común entre el editor y el suscriptor
- Una **clase que utilice los métodos de la interfaz**: el editor
- Una **clase que implemente** (rellene el contenido de) **los métodos de la interfaz**: el suscriptor (Hasta ahora solo hemos utilizado este punto, los otros dos nos lo daba la biblioteca de Android)

Veamos un ejemplo simple en el que crearemos un evento a la espera de que se procese algo. Este algo lo simularemos con un "sleep()" que haga dormirse al hilo (hago notar que esto es un ejemplo para ver que el evento se ejecutará en un tiempo que no sabemos, nunca hay que hacer esperar al hilo principal) y que al despertarse lance nuestro evento, notificando a todos los suscriptores a este evento.



Empezaremos definiendo una **interfaz** que he llamado "OnMiListener.java" (para que los asocies con el ejemplo del botón, podría haber llamado a esta clase "OnClickListener.java") con un método "onTodoProcesado()" (este podría haberlo llamado "onClick()") que además le paso una variable (podría haber sido "View"):

OnMiListener.java

```
public interface OnMiListener {
      public void onTodoProcesado(long tiempoTotal);
}
```

Una clase implementará los métodos del interfaz y así tendría algo que hacer cuando se todo fuera procesado. Antes vamos a crear el editor de eventos, que será la **clase que utilice los métodos de la interfaz.** La llamaremos "ClaseEditoraDeEventos.java" (en la asociación con el botón de antes, sería la clase "Button"). Para que sea simple, lo único que hará será esperar unos segundos aleatorios y cuando acabe llamará a nuestro método "onTodoProcesado()"; desde luego, para llamar a este método tiene que pasarse un objeto del tipo de la interfaz "OnMiListener.java" con el método ya implementado.

```java
ClaseEditoraDeEventos.java
public class ClaseEditoraDeEventos {

	public ClaseEditoraDeEventos(OnMiListener listener) {

		//Simula el tiempo aleatorio de descargar una imagen, al dormir unos milisegundos aleatorios al hilo en segundo plano
		long tiempoEnDescargar = (long) (Math.random() * 10);
		try {
			Thread.sleep(tiempoEnDescargar);
			listener.onTodoProcesado(tiempoEnDescargar);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

	}

}
```

Ya tenemos todo. Ahora nos queda lo fácil, lo que ya hemos hecho en otras ocasiones, que es **implementar los métodos de la interfaz**. Vamos a usar la clase "MainActivity.java" para poder ejecutar esta aplicación. Aquí tendremos que hacer lo de siempre, implementar al escuchador, el "OnMiListener"; implementar los métodos, en este caso "onTodoProcesado()"; y suscribirnos al evento pasándole el objeto con "this" al objeto del tipo "ClaseEditoraDeEventos" (también tendremos la opción de utilizar una clase anónima, como ya se vio).

```java
MainActivity.java
public class MainActivity extends Activity implements OnMiListener {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		ClaseEditoraDeEventos editor = new ClaseEditoraDeEventos(this);
	}

	@Override
	public void onTodoProcesado(long tiempoTotal) {
		Log.v("test", "Se ha procesado todo en este tiempo: " + tiempoTotal);
	}

}
```

Ya está. Si ejecutamos veremos cómo efectivamente el objeto suscrito recibe el evento y muestra por el LogCat el texto que hemos puesto:

```
-Se ha procesado todo en este tiempo: 5
```

¿Cómo se pasa un evento generado desde un Fragment a su Activity contenedora?

Es muy interesante saber cómo pasar eventos (y por tanto información) desde un Fragment a la Activity contenedora de este Fragment.

Se hace exactamente igual que hemos visto en la anterior pregunta. Esta vez la clase editora de eventos será el propio Fragment (es quien tiene que avisar cuando se lance el evento).

Solo nos faltan unas consideraciones de los eventos de los Fragments, en relación a su ciclo de vida y los eventos, que no hay que dejar nunca pasar:

- Solo podrá suscribirse la Activity a los eventos del Fragment cuando el Fragment exista (sino no tiene sentido). El mejor momento es hacerlo en el "**onAttach()**" (recuerdo del ciclo de vida del Fragment, que se ejecuta en cuanto el Fragment se ha adjuntado a la Acitvity contenedora); que para ello nos devuelve la Activity que contiene a este Fragment. Esta Activity tiene que implementar al 100% de nuestro interfaz "OnMiListener", por realizando un cast obtenemos la implementación. Fíjate que digo al 100% o dará

error, para que no se nos olvide existe manera elegante de comprobar si la Activity contenedora está implementando o no de la interfaz que necesita el Fragment: mediante un “try y catch”; esto nos recordará como desarrolladores que la tenemos que implementar, para que no se nos olvide, ya que nos dará error sino lo hacemos y nos mostrará nuestro mensaje recordándonos que tenemos que implementar de la interfaz.

```
private OnMiListener mListener;

@Override
public void onAttach(Activity activity) {
        super.onAttach(activity);

        try {
                mListener = (OnMiListener) activity;
        } catch (ClassCastException e) {
                throw new IllegalStateException("La clase " + activity.toString() + " debe implementar de la
interfaz " + OnMiListener.class.getName() + " del Fragment al que quiere escuchar");
        }

}
```

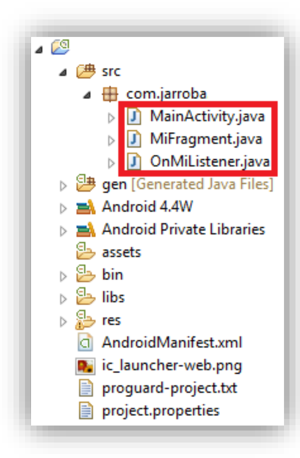
- Al contrario, cuando el Fragment deja de estar sobre la Activity contenedora, no puede haber eventos de esta (ya que no existirán). Por ello en el “**onDetach()**” (recuerdo del ciclo de vida, que se ejecuta quitarse el Fragment de la Activity contenedora) pondremos una implementación vacía (en el siguiente ejemplo he llamado “mListenerVacio”)

```
private static OnMiListener mListnerVacio = new OnMiListener() {
        @Override
        public void onTodoProcesado(long tiempoTotal) {
                //Evento vacío para que no haya problemas en algunos momentos del ciclo de vida del Fragment
        }
};

@Override
public void onDetach() {
        super.onDetach();
        mListener = mListnerVacio;
}
```

- Es bueno que al inicializar el Fragment la variable que guarda la implementación (he llamado en el código de abajo “mListener”), se inicie con la implementación vacía.

```
private OnMiListener mListener = mListnerVacio;
```



El resto es exactamente igual a como lo vimos anteriormente. La estructura del programa lo puedes ver a la derecha, reutilizaremos por completo el código de “OnMiListener.java” de la anterior pregunta.

Indicar que el resultado del programa será el mismo que en la pregunta anterior.

El código completo de “MiFragment.java” es (la variable “TAG_MI_ACTIVITY” veremos su significado en la siguiente pregunta):

MiFragment.java

```
public class MiFragment extends Fragment {

        public static final String TAG_MI_FRAGMENT = "IdentificadorDeMiFragment";

        private static OnMiListener mListnerVacio = new OnMiListener() {
                @Override
                public void onTodoProcesado(long tiempoTotal) {
                }
        };

        private OnMiListener mListener = mListnerVacio;

        public MiFragment() {
        }

        @Override
        public void onAttach(Activity activity) {
                super.onAttach(activity);
```

```
		try {
			mListener = (OnMiListener) activity;
		} catch (ClassCastException e) {
			throw new IllegalStateException("La clase " + activity.toString() + " debe 
implementar de la interfaz " + OnMiListener.class.getName() + " del Fragment al que quiere escuchar");
		}

	}

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		//Simula el tiempo aleatorio de descargar una imagen, al dormir unos milisegundos aleatorios 
al hilo en segundo plano
		long tiempoEnDescargar = (long) (Math.random() * 10);
		try {
			Thread.sleep(tiempoEnDescargar);
			mListener.onTodoProcesado(tiempoEnDescargar);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

	}

	@Override
	public void onDetach() {
		super.onDetach();
		mListener = mListnerVacio;
	}

}
```

En este ejemplo he utilizado un Fragment sin interfaz gráfica (en el ejemplo completo de más abajo podrás ver un ejemplo con interfaz gráfica, por si este de primeras no te queda del todo claro), para simplificar el código y explicar otras cosas además.

No extiendo más la respuesta a esta pregunta. Para que quede bien claro te recomiendo que hagas el ejemplo completo de más abajo que se llama: “Ejemplo de escuchar un evento (y pasar información) de un Fragment a la Activity contenedora”.

El código de “MainActivity.java” lo muestro en la siguiente pregunta para aprovechar a contar algo nuevo.

¿Cómo añado un Fragment sin interfaz gráfica a una Activity contenedora?

Si te fijas del ciclo de vida del Fragment –llamado “MiFragment.java”- de la anterior pregunta: no existe el método “onCreateView()” que mediante el método “inflate()” le proporciona la interfaz gráfica al usuario.

Podemos utilizar un Fragment sin interfaz gráfica para realizar operaciones y que el usuario no las vea, pero que aun así quede el código modular en Fragments. Esto es muy útil si por ejemplo en un Smartphone no necesitemos hacer cierta operación que sí necesite hacerse en un Tablet (una operación es algo sin interfaz gráfica y si no se necesita no va a estar ocupando memoria).

Reutilizo y completo el ejemplo de la anterior pregunta.

Para crear un Fragment sin interfaz gráfica necesariamente tenemos que añadir el Fragment a la Activity contenedora de manera dinámica, desde Java. Los pasos a seguir son para añadir el Fragment sin interfaz gráfica a la Activity contenedora son:

1. Obtener el interfaz para interactuar con Fragments dentro de la Activity con “**getFragmentManager()**”

```
FragmentManager fragmentManager = getFragmentManager();
```

2. Comenzaremos una nueva transacción al añadir (al usar “add()”) este Fragment con “**beginTransaction()**”

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

3. Obtendremos una **instancia del Fragment que queramos añadir**

```
MiFragment mifragment = new MiFragment();
```

4. Añadiremos con "**add()**" a la transacción, además le pondremos un identificador para si tenemos que volver a buscarlo en un futuro (lo suyo es que sea una variable global estática y final en el propio Fragment, para tener acceso desde sitios)

```
public static final String TAG_MI_FRAGMENT = "IdentificadorDeMiFragment";

fragmentTransaction.add(mifragment, TAG_MI_FRAGMENT);
```

5. Para terminar aceptamos todas las modificaciones (en este caso añadir un Fragment nuevo) con "**commit()**". En este momento el Fragment estará adjuntado a la Activity contenedora (se llamará al "OnAttach()" del Fragment y demás ciclo de vida)

```
fragmentTransaction.commit();
```

Una nota al importar bibliotecas: importa siempre los que empiecen por "android.app" como "android.app.FragmentManager" o "android.app.FragmentTransaction".

Y ya el código completo. Recuerdo que implementamos de "OnMiListener" para continuar con el anterior ejemplo.

MainActivity.java

```
public class MainActivity extends Activity implements OnMiListener {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		FragmentManager fragmentManager = getFragmentManager();
		FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

		MiFragment mifragment = new MiFragment();
		fragmentTransaction.add(mifragment, "IdentificadorDeMiFragment");
		fragmentTransaction.commit();
	}

	@Override
	public void onTodoProcesado(long tiempoTotal) {
		Log.v("test", "Se ha procesado todo en este tiempo: " + tiempoTotal);
	}

}
```

¿Para añadir un Fragment con interfaz gráfica de manera dinámica, es decir, desde Java se hace igual?

Sí, solo que cambiando el método add() con uno que tenga una View que sirva de contenedora para la interfaz gráfica (lo veremos en los siguientes temas con ejemplos). El tag es opcional, ya que podemos encontrar al Fragment recorriendo el árbol de Views.

```
public static final String TAG_MI_FRAGMENT = "IdentificadorDeMiFragment";

fragmentTransaction.add(R.id.viewContenedora, mifragment, TAG_MI_FRAGMENT);
```

¿Cómo se pasa un evento o información de la Activity contenedora a un Fragment que esta contiene?

Simplemente desde la Activity podremos llamar a métodos del otro Fragment.

Para ejemplificar esto hemos creado un ejemplo completo más abajo, llamado "Ejemplo de escuchar un evento (y pasar información) de un Fragment a otro"

Referencias:

- http://jarroba.com/programar-fragments-fragmentos-en-android/
- http://jarroba.com/fragments-fragmentos-en-android/
- http://developer.android.com/guide/components/fragments.html
- http://developer.android.com/training/basics/fragments/communicating.html

Ejemplo de escuchar un evento (y pasar información) de un Fragment a la Activity contenedora

Lo que haremos

Tendremos una Activity contenedora (para diferenciarlo pintaremos su fondo de rojo) que contiene un único Fragment (para diferenciarlo estará coloreado de verde claro). El Fragment tendrá un botón para que al ser pulsado ha de cambiar un texto de sobre Activity contenedora (por tanto fuera del Fragment).

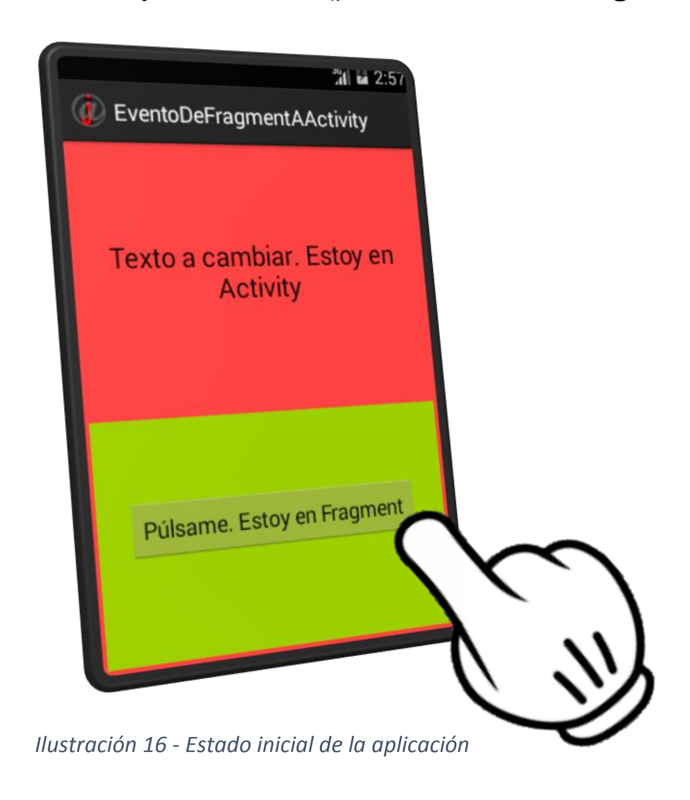
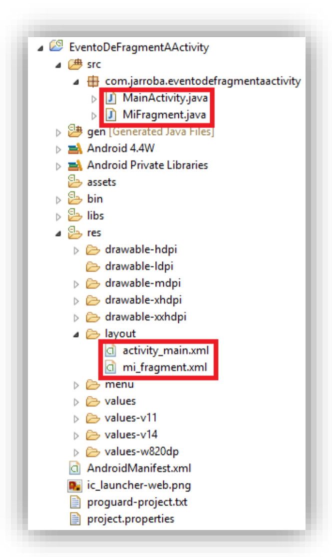


Ilustración 16 - Estado inicial de la aplicación



Ilustración 17 - Al pulsar el botón el texto de la Activity contenedora cambia



MainActivity.java

```
public class MainActivity extends Activity implements OnMiClickListener {

        private TextView texto;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                texto = (TextView) findViewById(R.id.textView$activity_main$texto);
        }

        @Override
        public void onClickEnMiBoton(View v) {
                texto.setText("Texto cambiado");
        }

}
```

MiFragment.java

```
public class MiFragment extends Fragment {

        public interface OnMiClickListener {
                public void onClickEnMiBoton(View v);
        }

        private static OnMiClickListener mListnerVacio = new OnMiClickListener() {
                @Override
                public void onClickEnMiBoton(View v) {
                }
        };

        private OnMiClickListener mListener = mListnerVacio;

        public MiFragment() {
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
                View rootView = inflater.inflate(R.layout.mi_fragment, container, false);

                Button boton = (Button) rootView.findViewById(R.id.button$mi_fragment$boton);
                boton.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                mListener.onClickEnMiBoton(v);
                        }
                });

                return rootView;
        }
```

```java
	@Override
	public void onAttach(Activity activity) {
		super.onAttach(activity);

		try {
			mListener = (OnMiClickListener) activity;
		} catch (ClassCastException e) {
			throw new IllegalStateException("La clase " + activity.toString() + " debe 
implementar de la interfaz " + OnMiClickListener.class.getName() + " del Fragment al que quiere escuchar");
		}

	}

	@Override
	public void onDetach() {
		super.onDetach();
		mListener = mListnerVacio;
	}

}
```

res/layout/activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_red_light"
    android:orientation="vertical"
    tools:context="com.jarroba.eventodefragmentafragment.MainActivity"
    tools:ignore="MergeRootFrame" >

    <TextView
        android:id="@+id/textView$activity_main$texto"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="center"
        android:text="Texto a cambiar. Estoy en Activity"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <fragment
        android:id="@+id/fragment$activity_main$miFragment"
        android:name="com.jarroba.eventodefragmentaactivity.MiFragment"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_margin="10dp"
        android:layout_weight="1"
        tools:layout="@layout/mi_fragment" />

</LinearLayout>
```

res/layout/mi_fragment.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light"
    tools:context="com.jarroba.eventodefragmentafragment.MainActivity$PlaceholderFragment" >

    <Button
        android:id="@+id/button$mi_fragment$boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="59dp"
        android:text="Púlsame. Estoy en Fragment" />

</RelativeLayout>
```

Ejemplo de escuchar un evento (y pasar información) de un Fragment a otro

Lo que haremos

Tendremos una Activity contenedora (para diferenciarlo pintaremos su fondo de rojo) con dos Fragments (para diferenciarlos los pintaremos uno de verde claro y el otro de verde oscuro). Uno de los Fragments con un botón y el otro con un texto que queremos cambiar al pulsar el botón.

Además, aprovechando que el evento tiene que pasar por la Activity contenedora, pondremos una imagen sobre la Activity contenedora que cambiará también al pulsar el botón.

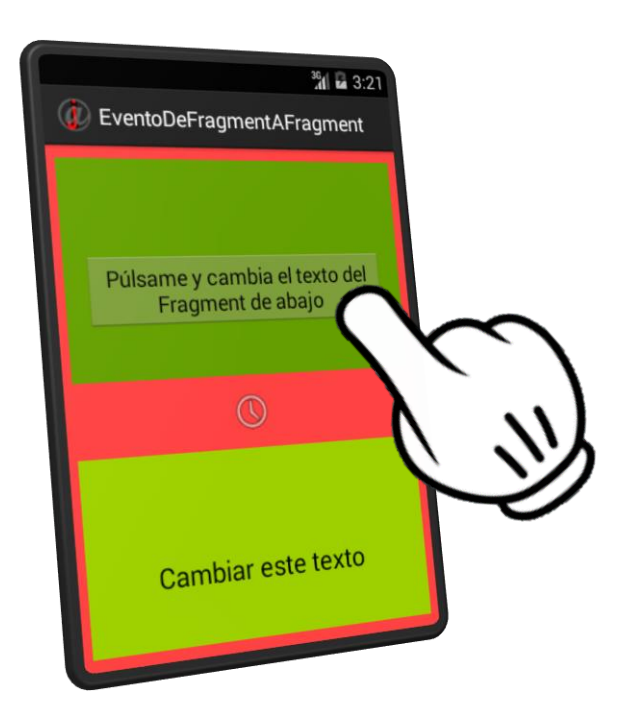


Ilustración 18 - Estado inicial de la aplicación

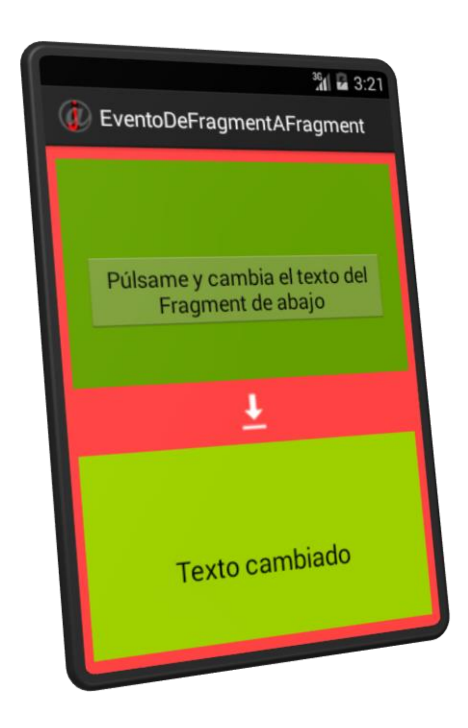
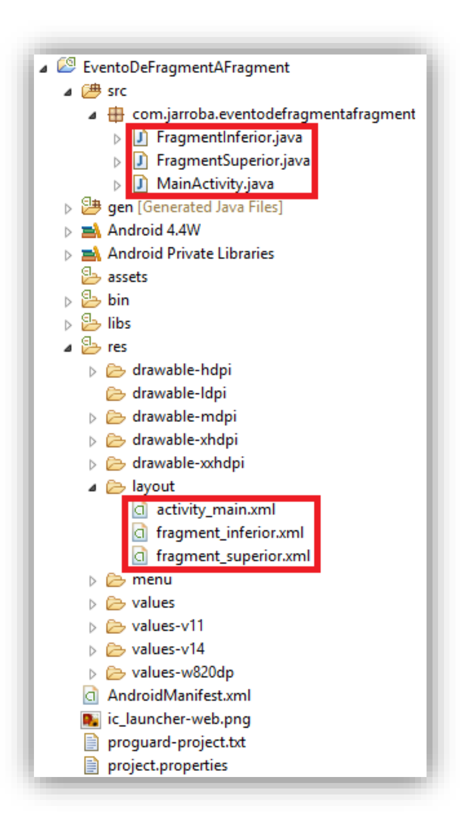


Ilustración 19 - Una vez pulsado el botón, se cambiará la imagen de la Activity contenedora y el texto del otro Fragment



FragmentInferior.java

```
public class FragmentInferior extends Fragment {

        private TextView textoCambiar;

        public FragmentInferior() {
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
                View rootView = inflater.inflate(R.layout.fragment_inferior, container, false);

                textoCambiar = (TextView)
rootView.findViewById(R.id.textView$fragment_inferior$textoCambiarFragmentInferior);

                return rootView;
        }

        public void comunicarmeConElFragment() {
                textoCambiar.setText("Texto cambiado");
        }

}
```

FragmentSuperior.java

```
public class FragmentSuperior extends Fragment {

        public interface OnMiClickListener {
                public void onClickEnMiBoton(View v);
        }

        private static OnMiClickListener mListnerVacio = new OnMiClickListener() {
                @Override
                public void onClickEnMiBoton(View v) {
                }
        };

        private OnMiClickListener mListener = mListnerVacio;

        public FragmentSuperior() {
        }
```

```java
	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_superior, container, false);

		Button boton = (Button)
rootView.findViewById(R.id.button$fragment_superior$botonDelFragmentSuperiorQueCambiaTexto);
		boton.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				mListener.onClickEnMiBoton(v);
			}
		});

		return rootView;
	}

	@Override
	public void onAttach(Activity activity) {
		super.onAttach(activity);

		try {
			mListener = (OnMiClickListener) activity;
		} catch (ClassCastException e) {
			throw new IllegalStateException("La clase " + activity.toString() + " debe
implementar de la interfaz " + OnMiClickListener.class.getName() + " del Fragment al que quiere escuchar");
		}

	}

	@Override
	public void onDetach() {
		super.onDetach();
		mListener = mListnerVacio;
	}

}
```

MainActivity.java

```java
public class MainActivity extends Activity implements OnMiClickListener {

	private ImageView imagen;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		imagen = (ImageView) findViewById(R.id.imageView$activity_main$imagenDeActivity);
	}

	@Override
	public void onClickEnMiBoton(View v) {
		imagen.setImageResource(android.R.drawable.stat_sys_download_done);

		FragmentInferior fInferior = (FragmentInferior)
getFragmentManager().findFragmentById(R.id.fragment$activity_main$fragmentInferior);
		fInferior.comunicarmeConElFragment();
	}

}
```

res/layout/activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_red_light"
    android:orientation="vertical"
    tools:context="com.jarroba.eventodefragmentafragment.MainActivity"
    tools:ignore="MergeRootFrame" >

    <fragment
        android:id="@+id/fragment$activity_main$fragmentSuperior"
        android:name="com.jarroba.eventodefragmentafragment.FragmentSuperior"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_weight="1"
        tools:layout="@layout/fragment_superior" />

    <ImageView
        android:id="@+id/imageView$activity_main$imagenDeActivity"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="0.32"
        android:contentDescription="Imagen de la Activity"
        android:src="@android:drawable/ic_menu_recent_history" />

    <fragment
        android:id="@+id/fragment$activity_main$fragmentInferior"
        android:name="com.jarroba.eventodefragmentafragment.FragmentInferior"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_weight="1"
        tools:layout="@layout/fragment_inferior" />
</LinearLayout>
```

res/layout/fragment_inferior.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light"
    tools:context="com.jarroba.eventodefragmentafragment.MainActivity$PlaceholderFragment" >

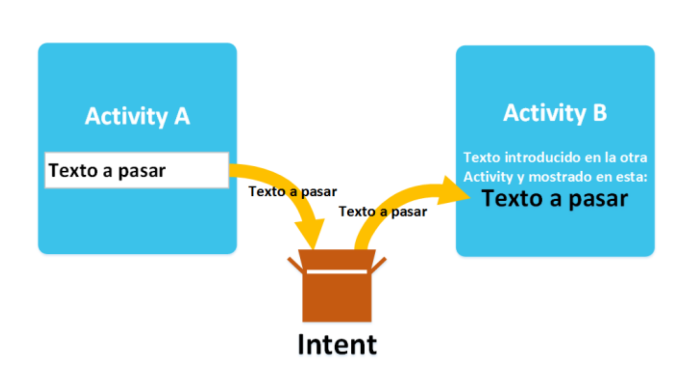
    <TextView
        android:id="@+id/textView$fragment_inferior$textoCambiarFragmentInferior"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="64dp"
        android:text="Cambiar este texto"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

res/layout/activity_main.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_dark"
    tools:context="com.jarroba.eventodefragmentafragment.MainActivity$PlaceholderFragment" >

    <Button
        android:id="@+id/button$fragment_superior$botonDelFragmentSuperiorQueCambiaTexto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="59dp"
        android:text="Púlsame y cambia el texto del Fragment de abajo" />
</RelativeLayout>
```

Intent



¿Qué es?

Unen componentes individuales en tiempo de ejecución. Es decir, son paquetes asíncronos (que contienen, por ejemplo, datos o acciones) que solicitan una acción a otro componente. Define un mensaje para activar un componente específico o un tipo de componente específico.

Dicho de otro modo, es una "intención" de hacer algo, como arrancar una Activity, tenemos la intención de arrancarla pero no la vamos a arrancar en el momento que se crea este Intent; se pasará a otro lado ésta intención para que ejecute su contenido, en este ejemplo iniciar una nueva Activity.

De este modo un Intent es un objeto que actúa como mensaje para realizar una petición de una acción de otro componente

¿Por qué se explica Intent en la navegación?

Porque es la pieza clave para navegar entre Activity y pasar datos entre estas

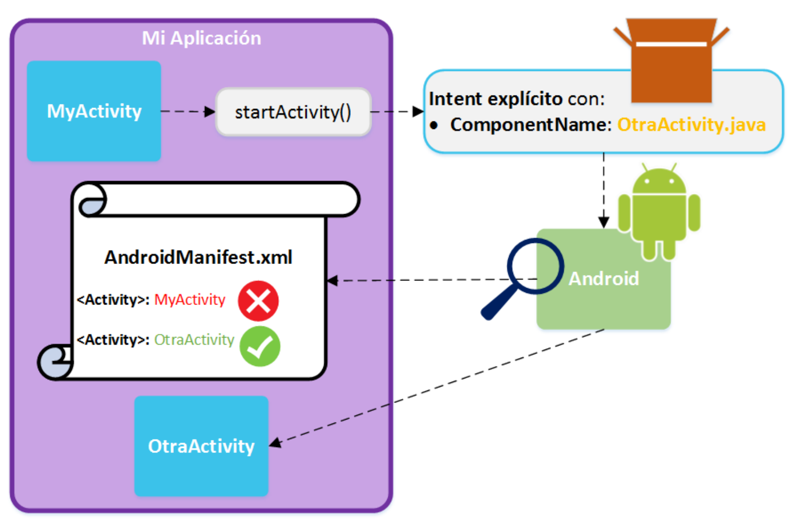
¿Dónde y qué componentes lo usan?

Pongo una recopilación de dónde se utiliza y qué componentes lo usan:

- Para iniciar una **Activity:**
  - Context.**startActivity**(): inicia una nueva Activity
  - Activity.**startActivityForResult**(): inicia una nueva Activity, además solicita que le devuelva resultados cuando finalice de cargar la nueva Activity a la que la inició.
- Para iniciar un **Service** (se verá en temas posteriores en profundidad):
  - Context.**startService**(): inicia un nuevo Service o entregar nuevas instrucciones a alguno en ejecución
  - Context.**bindService**(): para ligarse con un Service, creándolo si no existe
- Para enviar un **Broadcast** (se verá en temas posteriores en profundidad):
  - Context.**sendBroadcast**(): envía el Intent a todos los BroadcastReceivers interesados
  - Context.**sendOrderedBroadcast**(): Como el anterior, pero recibiendo datos desde el broadcast
  - Context.**sendStickyBroadcast**(): Como el primero, pero manteniendo el Intent aunque el broadcast se haya completado

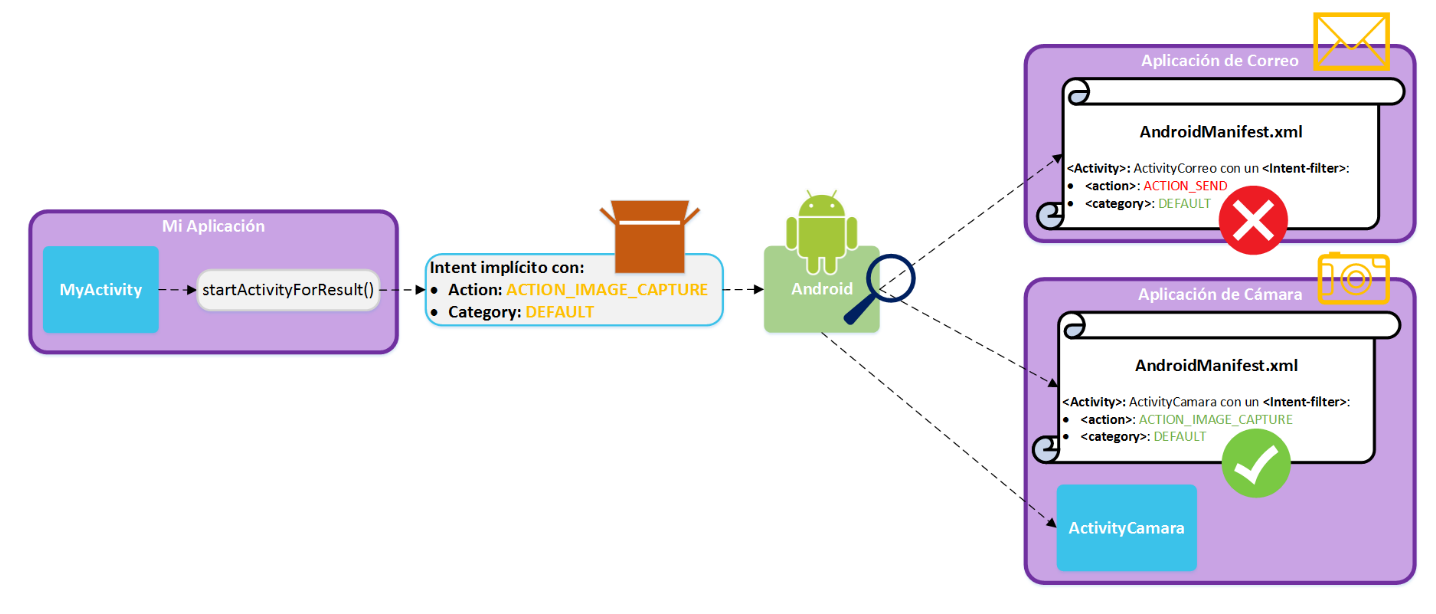
¿Tipos de Intent?

- **Explícito**: creado solo para el componente objetivo. Para el envío de mensajes y para instanciar clases de componentes de la misma aplicación. Ejemplo: arrancar una nueva Activity de la misma aplicación. Funciona del siguiente modo (mira la imagen mientras lees los siguientes puntos):



  1. Un componente (en la imagen de ejemplo un "MyActivity") **crea un Intent para iniciar otro de la misma aplicación** (en el ejemplo "OtraActivity").
  2. Se le **envía el Intent** que hemos creado al sistema operativo Android a través de un método para arrancar componentes, como startActivity()
  3. **Android buscará en el AndroidManifest.xml** de nuestra aplicación el componente con el nombre de la clase que queremos iniciar (en el ejemplo queremos iniciar la Activity llamada "OtraActivity"), no siendo elegibles ninguna con otro nombre salvo el pasado
  4. **Al encontrar el componente Android lo instanciará**

- **Implícito**: No tienen un objetivo claro. Son usados para activar componentes en otras aplicaciones. Android tiene que buscar el mejor componente que coincida con los parámetros del Intent. Para que la otra aplicación reciba el Intent, esta otra aplicación requiere de un Intent Filter configurado para que coincidan las estructuras asociadas (action, data, category), para así instanciar el componente adecuado recibido en el Intent. Ejemplo: desde la Activity de una App, se quiere llamar a otra Activity de otra aplicación que no es la nuestra, como la Activity que abre la cámara de fotos. Funciona así:



  1. Un componente (en la imagen de ejemplo un "MyActivity") **crea un Intent para iniciar otro de otra aplicación**. Aquí no conocemos como se llama el componente de la otra aplicación, por lo que tenemos que usar unos filtros comunes (lo podemos encontrar en la documentación de Android en http://developer.android.com/reference/android/content/Intent.html) para comunicarnos de manera efectiva. Para esto utilizamos Action, Category y Data (en el ejemplo utilizamos solo Action con el valor "ACTION_IMAGE_CAPTURE" y Category con el valor "DEFAULT"). Indicar que Category ya estará añadido por defecto con "DEFAULT" y no hará falta que lo añadamos nosotros (Hago notar que lo que se pone en Action y en Category son constantes con un String creado por

Android para que sea común en todas las aplicaciones. Otra cosa, en los ejemplo para que quede más limpio pongo por ejemplo solo “DEFAULT”, para que funcione habría que poner “android.intent.category.DEFAULT”).

2. Se le **envía** a Android **el Intent** que hemos creado a través de un método para arrancar componentes. Aquí es posible que nos interese que el otro componente de la otra aplicación nos devuelva un resultado cuando acabe (por ejemplo queremos usar la foto que haga la aplicación de la cámara al volver a nuestra Activity, que en el ejemplo he llamado “MyActivity”), por ello estará bien usar startActivityForResult()

3. **Android buscará en todos los AndroidManifest.xml de todas las aplicaciones todos los en los <intent-filter> de todos los componentes** de nuestro dispositivo, hasta encontrar todas las coincidencias con los criterios de búsqueda puestos en Action, Category y Data (en el caso del ejemplo queremos que coincida el Action y el Category del Intent con el <action> y <category> de los <intent-filter> de los AndroidManifest.xml). Si creamos nosotros la otra aplicación, es importante que definamos en el AndroidManifest.xml el <category> como “DEFAULT” para recibir los Intents implícitos (es justo lo que estás pensando, puedes utilizar esto para comunicar varias aplicaciones de tu creación).

4. **Si** solo **hay** una **coincidencia Android lanzará ese** único **componente** de esa otra aplicación. Si hay varias coincidencias Android dejará elegir al usuario cuál utilizar. Si no hay ninguna, Android notificará al usuario.

¿Cómo creo una Activity, envió y recibo datos entre ellas?

Para enviar un dato a otra Activity e iniciarla lo primero que hay que hacer es crear un Intent en la Activity que iniciará a la otra.

- **Enviar datos**: Los datos a pasar se llaman “Extras” por lo que llamaremos al método “putExtra()” que nos pide dos parámetros, una clave para identificar el valor, y este valor -que es el dato que queremos pasar- identificado por la clave. El valor puede ser cualquier tipo primitivo (int, String, float, etc) y algunos complejos (veremos en otra pregunta como pasar objetos definidos por nosotros). Es recomendable que la clave sea única y que sea una variable global publica final y estática, para poder acceder a ella desde otras clases.

```
public final static String CLAVE_EXTRA_PASAR = "claveDelDatoAPasarALaOtraActivity";

Intent intencion = new Intent(getApplicationContext(),ActivityQueQuieroIniciar.class);

intencion.putExtra(CLAVE_EXTRA_PASAR, textoAPasar);

startActivity(intencion);
```

- **Recibir datos**: Cuando se inicie la nueva Activity, para recibir los “Extras”, es decir los datos que nos manda la otra Activity, solo tenemos que obtener el Bundle con los datos con “getIntent().getExtras()” y ya de ahí obtener el valor primitivo que hayamos pasado. Por ejemplo, para obtener un “String” con “getString()” o un “int” con “getInt()” (para obtener objetos definidos por nosotros lo veremos en otra pregunta). A este método para obtener el valor hay que pasarle la clave que antes definimos, para obtener el valor específico que queremos y un segundo parámetro definirá el valor por defecto (no es obligatorio, pero lo recomiendo para que por si no nos devuelve ningún dato que no nos de problemas y obtengamos un dato controlable).

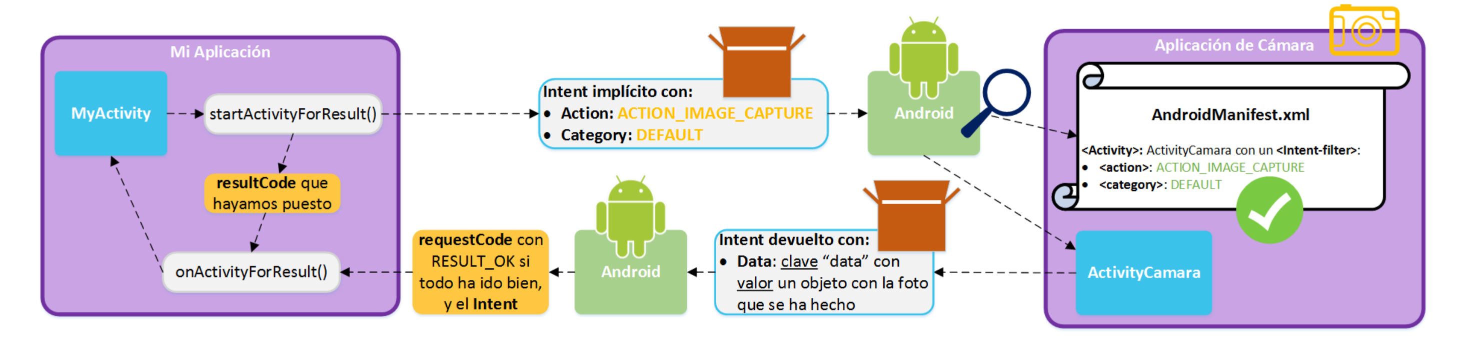
```
Bundle bundle = getIntent().getExtras();

String textoRecibido = bundle.getString(LaAnteriorActivity.CLAVE_EXTRA_PASAR, "texto
por si la clave no tiene asignado un valor");
```

¿Cómo iniciar una Activity que no sea de mi aplicación?

Si utilizamos un Intent implícito, esto es abrir una Activity externa a nuestra aplicación. Seguramente nos interese detectar cuando volvamos a nuestra Activity y hacer algo con los datos que nos ha devuelto la Activity externa a nuestra aplicación.

Supongamos el siguiente ejemplo: tenemos una Activity con un botón que al ser pulsado lanza la Activity de la cámara que está en otra aplicación de terceros. Al hacer la foto la aplicación del tercero se cerrará y volveremos a nuestra aplicación, a nuestra Activiy que lanzó la Activity externa. Para detectar el momento en que volvemos y para hacer algo con la foto realizada con la aplicación externa tenemos que rellenar el método onActivityforResults().



Para ello **creamos un Intent implícito con una acción** (el siguiente código muestra que se crea un Intent con una acción "MediaStore.ACTION_IMAGE_CAPTURE", indicando a Android que queremos lanzar una aplicación que tenga alguna Activity capaz de captura una imagen). Para lanzar a la Activity utilizaremos startActivityForResult(), al que pasaremos el Intent implícito, y un código (requestCode) que nos servirá a nosotros para diferenciar que hacer dentro del onActivityforResults().

```
private static final int REQUESTCODE_CAPTURA_DE_IMAGEN = 1;

Intent miIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

if (miIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(miIntent, REQUESTCODE_CAPTURA_DE_IMAGEN);
}
```

Si hubiera más de una aplicación de cámara instalada en nuestro dispositivo, Android dará a elegir al usuario. A nosotros como desarrolladores nos da igual que aplicación haga la foto, solo queremos la foto. Lo que sí nos interesa es saber que haya al menos una aplicación que pueda hacer fotos. En el anterior pedazo de código, el "if" comprueba que al menos una Activity externa pueda utilizar nuestro Intent, que pueda hacer fotos.

Ya se ha abierto la otra aplicación, no tenemos su control de nada que pase en ella. Nos toca esperar a que termine y volvamos a nuestra Activity.

El método **onActivityforResults()** será llamado al volver a nuestra Activity siempre que ya existiera previamente (se llamará antes que a onResume()); dicho de otra manera, se llamará al volver de otra aplicación que no es la nuestra. Además devuelve una serie de parámetros:

- **requestCode** (código de respuesta): un código que definimos nosotros, para informarnos a nosotros mismos dentro startActivityforResults() que es lo que se ha hecho. Por si la Activity de terceros pudiera hacer varias cosas, diferenciar cual se ha hecho.
- **resultCode** (código de solicitud): este código es importante, ya que nos informa de varios sucesos. Aunque aquí voy a explicar todos, en la mayoría de los casos bastará con comprobar el primero:
    - **Activity.RESULT_OK**: si se ha realizado todo correctamente
    - Activity.RESULT_CANCELED: si la Activity de terceros no devuelve ningún resultado o se fallado mientras realizaba la operación
    - Activity.RESULT_FIRST_USER: Si se vuelve con resultados definidos por el usuario
- **Intent:** Intent que nos devuelve la Activity que externa a nuestra Activity

El siguiente código muestra un ejemplo de cómo implementar el método onActivityResult(). Utilizaremos el parámetro resultCode para comprobar si todo ha ido bien (si es RESULT_OK). Luego diferenciaremos con un switch que requestCode le está llegando (el que hemos pasado nosotros antes en el startActivityForResult()) para hacer una cosa u otra dependiendo de qué nos llegue. Ya nos queda utilizar el Intent con los datos que nos han pasado, para la foto vendrá en un Extra con la clave "data" (esto lo sabemos por la documentación).

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
        if (resultCode == Activity.RESULT_OK) {
                switch (requestCode) {
                        case REQUESTCODE_CAPTURA_DE_IMAGEN: {
                                Bundle extras = datos.getExtras();
                                Bitmap imagenBitmap = (Bitmap) extras.get("data");
                                break;
                        }
                }
        }
        super.onActivityResult(requestCode, resultCode, intent);
}
```

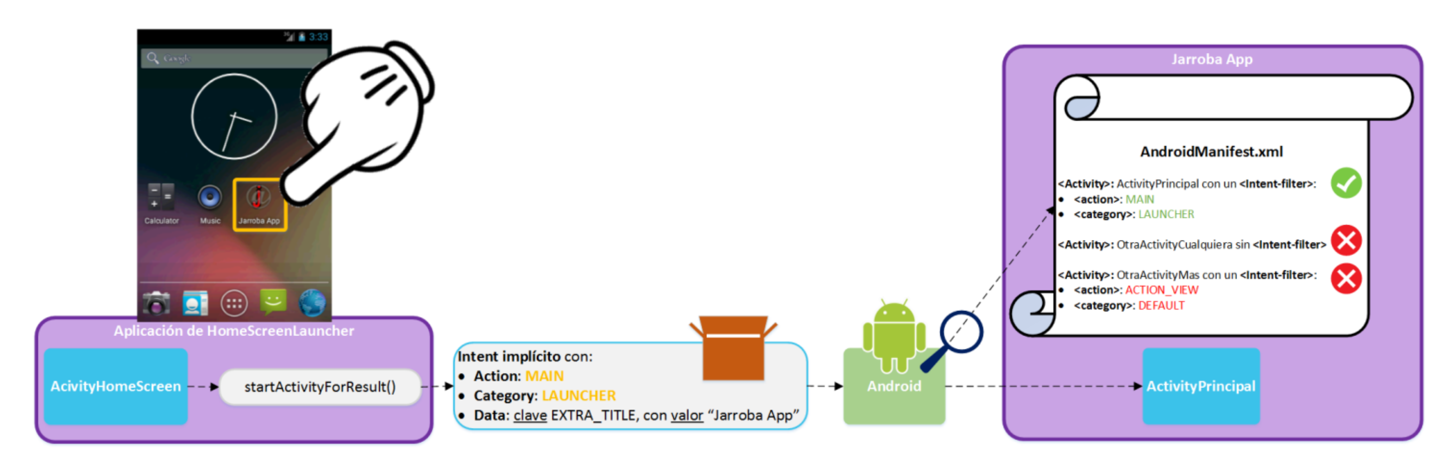
También podremos utilizar getData() del Intent devuelto, si por ejemplo lo que nos devuelve es una Uri (esto de la Uri lo veremos en temas posteriores). Aunque también nos podrá devolver otros datos en los "extras", para usarlos tendremos que ver en la documentación que nos está devolviendo.

Si todavía te quedaran dudas o quieres practicar, tienes el ejemplo completo un poco más adelante.

¿Por qué la Activity que arranca mi aplicación tiene un "action" y una "category" en el "AndroidManifest.xml"?

Como hemos explicado anteriormente, para que otra aplicación nos lance la Activity principal. Antes de que elimines esta action y este category, al pensar en que "ninguna aplicación va a ejecutar mi Activity" o "no quiero que ninguna aplicación inicie mi Activity" has de saber el por qué está ahí eso.

Siempre va a existir una aplicación que inicie tu aplicación, por tanto tu Activity, y esta es el escritorio de Android. Dicha aplicación se llama "HomeScreenLauncher" y viene instalado en todos los sistemas operativos de Android. Es la aplicación que suelen cambiar las casas que fabrican dispositivos para que se adecuen a su marca, o también la que suelen hacer varios desarrolladores para modificarla y que sea diferente (¿Quién no ha querido un escritorio de Android que sea igual a Windows 95? Es broma, ¿me sigues no?).



Lo que ocurre cuando pulsamos un icono del escritorio de la aplicación de "HomeScreenLauncher", realmente estamos pulsando un <Button> sobre una Activity que se llama "ActivityHomeScreen". Ejecuta el método startActivityForResult() al que le pasa un Intent implícito con un "Action" que contiene "MAIN" y una "Category" con "LAUNCHER", además de un "extra" con la clave "EXTRA_TITLE" con el valor del nombre de la aplicación. De este modo Android sabe que Activity abrir y evidentemente coincidirá el "Action" y el "Category" con la única Activity de nuestra aplicación que tenga estos dos, por lo que será la que arranque la aplicación. Es decir, hace lo mismo que cualquier otro Intent implícito. De ahí que sea esta Activity "ActivityHomeScreen" la que inicia nuestra aplicación cuando pulsamos sobre un icono del escritorio de Android.

¿Cómo pasar un objeto completo como Extra de un Intent?

Muy sencillo. Supongamos que queremos pasar un objeto como el siguiente, que he llamado "ObjetoComplejo.java". Lo primero que hay que hacerle es que implemente de Serializable. Al implementar de Serializable tiene que tener una variable global llamada serialVersionUID que sea de tipo "long".

¿Cómo genero automáticamente el serialVersionUID de una implementación Serializable?

Al implementar la clase de Serializable, Eclipse nos marcará de amarillo el nombre de la clase. Al poner el cursor encima nos aparecerá la opción "Add generated serial version ID" que nos creará automáticamente una serial. Nos aparecerá una variable como la siguiente:
private static final long serialVersionUID = 4033861841342628493L;

Ya solo nos queda crear nuestro objeto de la manera normal que hemos hecho siempre en Java. En el siguiente código pongo un ejemplo de que tenga dentro un String, un List de objetos y un objeto (reutilizo ObjetoComplejo para el ejemplo, podría ser cualquier otro objeto). Lo dicho, será un objeto normal con sus variables y métodos.

```
public class ObjetoComplejo implements Serializable {
        private static final long serialVersionUID = 4033861841342628493L;

        public String unDatoDeTexto = "";
        public List<ObjetoComplejo> unaLista = new ArrayList<ObjetoComplejo>();
        public ObjetoComplejo oc = new ObjetoComplejo();
}
```

Ahora bien, para enviar y recibir este objeto entre Activities:

- **Enviar datos**: Tengo mi objeto normal de Java que quiero pasar, que podré hacer con él lo que quiera antes y después. Simplemente lo pasamos al método putExtra() con una clave para reconocerlo y el objeto que quiera (por sobrecarga de métodos, el objeto que implementa de Serializable entrará en el método putExtra()).

```
public final static String CLAVE_EXTRA_PASAR = "claveDelDatoAPasarALaOtraActivity";

Intent intencion = new Intent(getApplicationContext(), ActivityQueQuieroIniciar.class);

ObjComplejo miObjetoPasar = new ObjetoComplejo();
intencion.putExtra(CLAVE_EXTRA_PASAR, miObjPasar);

startActivity(intencion);
```

- **Recibir datos**: Para recibirlo simplemente obtendremos con getSerializable() y le haremos un cast (convertir un objeto de un tipo en el tipo de objeto que nosotros queramos). Ya podremos utilizarlo en la otra Activity.

```
Bundle bundle = getIntent().getExtras();

ObjetoComplejo miObjRecibir  = (ObjetoComplejo) bundle.getSerializable(LaAnteriorActivity.CLAVE_EXTRA_PASAR);
```

Referencias:

- http://developer.android.com/guide/components/intents-filters.html
- http://developer.android.com/reference/android/content/Intent.html
- http://developer.android.com/training/basics/intents/result.html
- http://developer.android.com/guide/topics/appwidgets/host.html

Ejemplo de ir desde una Activity a otra dentro de la misma aplicación

Lo que haremos

Lo que escribamos en un cuadro editable (EditText) será pasado a una nueva Activity cuando pulsemos un botón (Como recordatorio: al crear otra Activity nos tenemos que acordar de declararla en el “AndroidManifest.xml”).

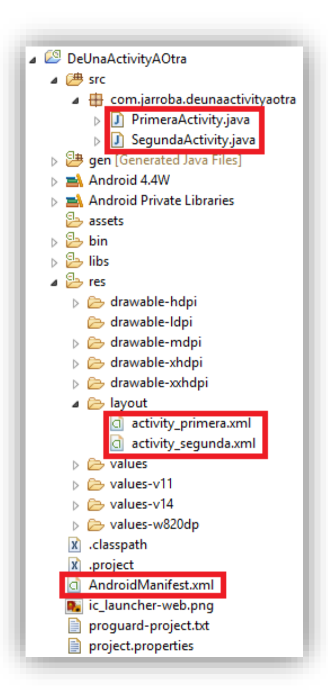
Para hacer este ejemplo utilizaremos un Intent explícito.



Ilustración 20 - Activity Primera donde escribiremos lo que queremos pasar a la nueva Activity y pulsaremos el botón para iniciar la nueve Activity



Ilustración 21 – Cuando se inicie la Activity Segunda recogerá el valor pasado y lo mostrará



PrimeraActivity.java

```
public class PrimeraActivity extends Activity {

	public final static String CLAVE_EXTRA_PASAR = "claveDelDatoAPasarALaOtraActivity";

	private EditText et;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_primera);

		et = (EditText) findViewById(R.id.editText$activity_primera$textoPasar);

		Button bt = (Button) findViewById(R.id.button$activity_primera$botonIniciarActivity);
		bt.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				String textoAPasar = et.getText().toString();

				Intent intencion = new Intent(getApplicationContext(), SegundaActivity.class);
				intencion.putExtra(CLAVE_EXTRA_PASAR, textoAPasar);
				startActivity(intencion);
			}
		});
	}
}
```

SegundaActivity.java

```
public class SegundaActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_segunda);

		Bundle bundle = getIntent().getExtras();
		String textoRecibido = bundle.getString(PrimeraActivity.CLAVE_EXTRA_PASAR, "texto por si la
clave no tiene asignado un valor");

		TextView tvDatoRecibido = (TextView)
findViewById(R.id.textView$activity_segunda$textoPasado);
		tvDatoRecibido.setText(textoRecibido);
	}
}
```

res/layout/activity_primera.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".PrimeraActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACTIVITY PRIMERA"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText$activity_primera$textoPasar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint=""
        android:text="Texto a pasar a la otra Activity" >
        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button$activity_primera$botonIniciarActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Iniciar una nueva Activity" />
</LinearLayout>
```

res/layout/activity_segunda.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_dark"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".SegundaActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACTIVITY SEGUNDA\n\nEl dato pasado es:"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/textView$activity_segunda$textoPasado"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@android:color/holo_red_light"
        android:text="xxxx"
        android:textAppearance="?android:attr/textAppearanceSmall" />
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.deunaactivityaotra"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".PrimeraActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="SegundaActivity"></activity>
    </application>

</manifest>
```

Ejemplo de ir desde una Activity a otra fuera de nuestra aplicación y volver a la anterior

Lo que haremos

Nuestra Activity tendrá un botón que abrirá el Activity de una aplicación de cámara instalada en nuestro dispositivo (Activity externa), al hacer la foto volveremos a nuestra Activity y dibujaremos la foto en un ImageView.

Nota: Las imágenes de este ejemplo están hechas con el emulador de Android. El emulador simula lo que ve la cámara con un fondo de cuadrados blancos y negros con un cuadrado encima que cambia de color (de ahí que en estas imágenes se vean cuadrados en vez de una foto real). Si lo probamos en un dispositivo real o con webcam, veremos una imagen real.

No hará falta que declaremos nada en el "AndroidMainfest.xml", puesto que se abre una Activity de otra aplicación (deberá estar declarada, por quien desarrollo la otra aplicación, en el "AndroidManifest.xml" de la otra aplicación). Por la misma razón, no harán falta declarar permisos de cámara, ya que nosotros utilizaremos la otra aplicación, donde deben estar declarados. Si fuésemos nosotros quienes desarrolláramos la aplicación de cámara, sí que tendríamos que declarar los permisos apropiados.

Si hubiera más de una aplicación de cámara instalada en nuestro dispositivo, Android dará a elegir al usuario. A nosotros como desarrolladores nos da igual que aplicación haga la foto, solo queremos la foto. Lo que sí nos interesa es saber que haya al menos una aplicación que pueda hacer fotos.

Para hacer este ejemplo utilizaremos un Intent implícito.



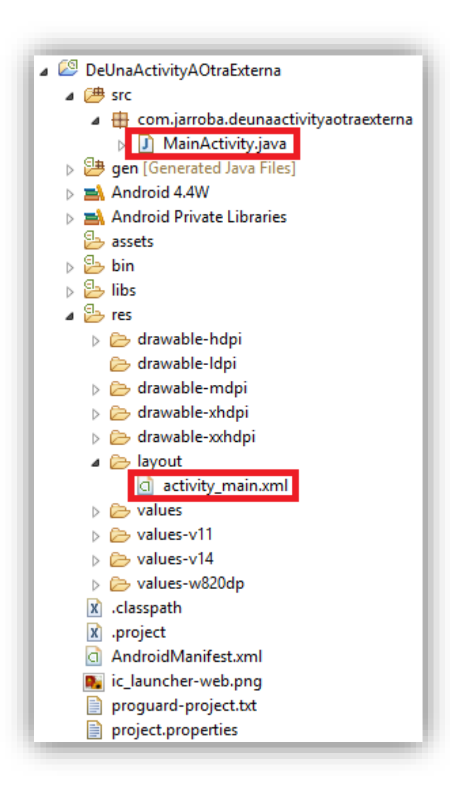
Ilustración 22 - El usuario tiene un botón que lanza una aplicación de cámara (lanza un Activity de la aplicación de la cámara)



Ilustración 23 - Se abre la cámara, el usuario hace una foto. Entonces, la aplicación de la cámara se cierra y volvemos a nuestra aplicación



Ilustración 24 - Al volver a nuestra aplicación, volveremso a la Activity que lanzó la cámara, por lo que entrará por su onActivityResult() con la foto. Teniendo la imagen solo nos queda dibujarla



MainActivity.java

```
public class MainActivity extends Activity {

	private ImageView imageViewFotografia;

	private static final int CODE_CAPTURA_DE_IMAGEN = 1;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		imageViewFotografia = (ImageView) findViewById(R.id.imageView$activity_main$ponerAquiFoto);
		final Intent miIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

		//Comprobamos que exista alguna Activity externa que pueda controlar esta petición
		if (miIntent.resolveActivity(getPackageManager()) != null) {
			Button botonHacerFoto = (Button) findViewById(R.id.button$activity_main$hacerFoto);
			botonHacerFoto.setOnClickListener(new OnClickListener() {
				@Override
				public void onClick(View v) {
					startActivityForResult(miIntent, CODE_CAPTURA_DE_IMAGEN);
				}
			});
		}
	}

	@Override
	protected void onActivityResult(int codigoDeSolicitud, int codigoDeResultado, Intent datos) {
		if (codigoDeResultado == Activity.RESULT_OK) {
			switch (codigoDeSolicitud) {
				case CODE_CAPTURA_DE_IMAGEN: {
					Bundle extras = datos.getExtras();
					Bitmap imagenBitmap = (Bitmap) extras.get("data");
					imageViewFotografia.setImageBitmap(imagenBitmap);
					break;
				}
			}
		}
		super.onActivityResult(codigoDeSolicitud, codigoDeResultado, datos);
	}

}
```

res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.deunaactivityaotraexterna.MainActivity" >

    <Button
        android:id="@+id/button$activity_main$hacerFoto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Abrir Activity de Cámara" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Imagen obtenida al volver de la Activity de la Cámara:"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <ImageView
        android:id="@+id/imageView$activity_main$ponerAquiFoto"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:src="@android:drawable/ic_menu_gallery"
        android:contentDescription="Imagen devuelta por la cámara"/>

</LinearLayout>
```

Navegación con Fragments

¿Qué tipos de Fragments hay? ¿Puedo cambiar cualquier tipo de Fragment desde Java?

Existen dos tipos de Fragments:

- **Estático o final**: se declara un fichero XML de la carpeta “layout” con las etiquetas <fragment>; donde se declara un atributo “android:name”, con nombre del paquete seguido de la clase Java que herede de Fragment, la cual le da funcionalidad. Este Fragment lo instancia Android en el momento en que carga el diseño de la Activity que lo contiene. Este Fragment va siempre pegado al diseño de la Activity contenedora. Es importante recordar que un Fragment estático no podrá ser nunca eliminado o sustituido desde Java, de intentarlo nos dará error. Para cambiar este Fragment habrá que sustituir completamente a la Activity (y con ella se irá el Fragment estático).

```
<fragment
    android:name="com.jarroba.miApp.MiClaseQueHeredaDeFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- **Dinámico**: se declara e instancia desde código Java, y se asocia a un ViewGroup (Se recomienda el uso de FrameLayout). Éste sí que se podrá ser añadido, eliminado o sustituido desde Java por otro Fragment u otro contenido.

```
MiClaseQueHeredaDeFragment miFragment = new MiClaseQueHeredaDeFragment();

getFragmentManager().beginTransaction()
        .add(R.id.frameLayoutDondePonerElFragmentDinamico, miFragment)
        .commit();
```

¿Puedo visualizar en el editor gráfico de diseños cómo quedaría un ejemplo del Fragment?

Claro que sí. Para facilitar la vida al desarrollador, como vimos podemos, podemos utilizar “tools:layout” para cargar el diseño que queramos y ver cómo queda.

```
<fragment
    android:name="com.jarroba.miApp.MiClaseQueHeredaDeFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout="@layout/layout_del_fragment_estatico" />
```

No puedo terminar la contestación de esta pregunta sin recordar que: esto es para desarrolladores, para el editor gráfico de diseños, el usuario nunca verá esto. Como recordatorio, para asociar un diseño a un Fragment se hace desde la clase que hereda de Fragment, en la parte del ciclo de vida del Fragment en onCreateView().

¿Cómo puedo añadir, modificar y eliminar Fragment dinámico?

Muy fácil. Lo voy a explicar con detalle. Podrás ver que estos pasos se resumen rápido. Antes voy a poner los requisitos que necesitamos:

1. Tener un diseño en una carpeta Layout XML con un **ViewGroup** como FrameLayout, **para colocar el Fragment dinámico** como:

```
<FrameLayout
    android:id="@+id/frameLayoutDondePonerElFragmentDinamico"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. Tener una **clase** Java **que herede de Fragment** como:

```
public class MiClaseQueHeredaDeFragment extends Fragment {
    //Código de la clase. Como el contructor vacío, la asociación del diseño en el onCreateView(), etc
}
```

Ahora los pasos para realizar cambios de un Fragment:

3. **Instanciar** u obtener la instancia del **Fragment** (de una clase Java que herede de Fragment)

```
MiClaseQueHeredaDeFragment miFragment = new MiClaseQueHeredaDeFragment();
```

4. **Obtener FragmentManager** (Interfaz para interactuar con un objeto de tipo Fragment en una Activity):
    a. Si vas a programar en Android 3.0 en adelante, **importaremos "android.app"**:

    ```
    FragmentManager fm = getFragmentManager();
    ```

    b. Si vas a programar en versiones más antiguas a Android 3.0, importaremos "android.support.v4.app" (intenta evitar el uso de esta biblioteca, ya no tiene soporte. Y sobre todo no mezcles bibliotecas bajo ningún motivo, o tendrás errores):

    ```
    FragmentManager fm = getSupportFragmentManager();
    ```

5. **Obtener FragmentTransaction** (Las transacciones de Fragments funcionan como si fueran una lista de acciones que hay que ejecutar en algún momento. Esto es, vamos poniendo todo lo que queremos que haga un Fragment al hacer la transición, como cambiar un Fragment por otro, añadir animaciones, que se guarde la pila de Fragments, etc; y cuando estemos contentos con lo que tiene que hacer ejecutaremos todo junto con commit()):

```
FragmentTransaction ft = fm.beginTransaction();
```

6. **Apilar las transiciones.** Cada vez que llamemos a algún método de estos, se apilará y ejecutará en orden de llamada a cada método cuando hagamos commit(). Ninguno de estos métodos se ejecuta en el instante de llamarse, solo cuando se llama al método commit(). Algunos que podremos usar son:
    a. **Añadir** un Fragment dinámico con interfaz gráfica a un ViewGroup.

    ```
    ft.add(R.id.frameLayoutDondePonerElFragmentDinamico, miFragment);
    ```

    b. **Eliminar** un Fragment dinámico existente de un ViewGroup.

    ```
    ft.remove(miFragment);
    ```

    c. **Remplazar** realmente lo que hace es eliminar y después añadir (ejecuta los dos métodos anteriores). Si no existiera un Fragment anterior no pasa nada, simplemente lo añade.

    ```
    ft.replace(R.id.frameLayoutDondePonerElFragmentDinamico, miFragment);
    ```

    d. **Añadir un Fragment dinámico sin interfaz gráfica.** Puede interesarnos hacer un Fragment que no tenga nada que ver el usuario, como lógica oculta. Es muy interesante para realizar operaciones ocultas al usuario y reutilizar el código de una manera muy sencilla. Es importante añadir un tag para poder identificar al Fragment desde otra parte de nuestro código; pues no se puede obtener del diseño, ya que no está asociado a ninguno, es solo lógica.

    ```
    ft.add(miFragment, "Nombre del tag");
    ```

    e. **Guardar estado actual** en la pila de atrás. Con esto al pulsar el botón de "atrás" del dispositivo, des-apilaremos antes el Fragment que a la Activity contenedora, mostrando el Fragment anterior al commit.

    ```
    ft.addToBackStack("Nombre opcional para este estado de la pila de atrás");
    ```

    f. Animar la transición con varios efectos que podemos encontrar dentro de la clase FragmentTransaction.

    ```
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ```

7. Hacer un **commit** para que los cambios de las transiciones surtan efecto (guiño a Git):

```
ft.commit();
```

Todos los pasos anteriores están muy bien para aprender. Cuando tengas un poco de práctica, y gracias a que todos los métodos de FragmentTransaction devuelven el mismo objeto FragmentTransaction, podremos reducir el código anterior a algo tan elegante como:

```
getFragmentManager().beginTransaction()
        .replace(R.id.frameLayoutDondePonerElFragmentDinamico, miElegido)
        .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
        .addToBackStack("Nombre opcional para este estado de la pila de atrás")
        .commit();
```

Referencias:

- http://developer.android.com/guide/components/fragments.html
- http://developer.android.com/reference/android/app/FragmentManager.html
- http://developer.android.com/reference/android/app/FragmentTransaction.html

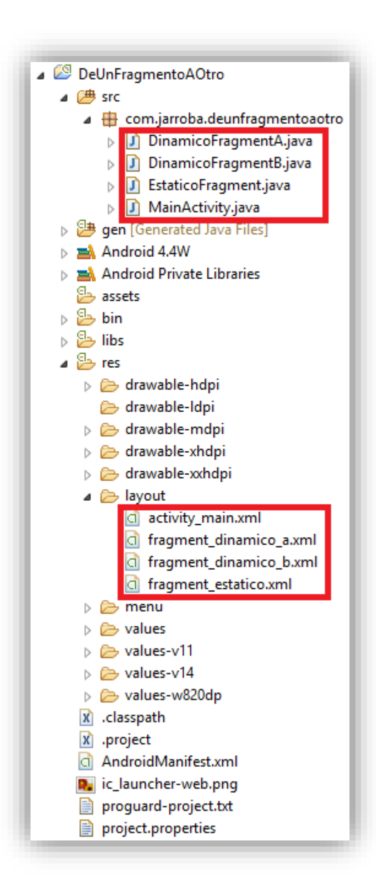
Ejemplo de cambiar de un Fragment a otro en diferentes dispositivos

Lo que haremos

Tendremos una aplicación con un Fragment estático en la parte superior. Este Fragment estático tendrá un botón para permutar entre dos Fragments dinámicos que quedarán ubicados justo debajo del estático. Es decir, al cargar la aplicación aparecerá el Fragment dinámico A; luego, cada vez que se pulse el botón si se veía el Fragment dinámico A pondremos el B, y viceversa.



Ilustración 25 - Cada vez que pulsemos el botón del Fragment estático, cambiaremos entre el Fragment dinámico A y B



DinamicoFragmentA.java

```
public class DinamicoFragmentA extends Fragment {

	public DinamicoFragmentA() {}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_dinamico_a, container, false);
		return rootView;
	}
}
```

DinamicoFragmentB.java

```
public class DinamicoFragmentB extends Fragment {

	public DinamicoFragmentB() {}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_dinamico_b, container, false);
		return rootView;
	}
}
```

EstaticoFragment.java

```
public class EstaticoFragment extends Fragment {

	public interface OnMiClickListener {
		public void onClickCambiarFragmentDinamico();
	}

	private static OnMiClickListener mListnerVacio = new OnMiClickListener() {
		@Override
		public void onClickCambiarFragmentDinamico() {}
	};

	private OnMiClickListener mListener = mListnerVacio;

	public EstaticoFragment() {}
```

```java
	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.fragment_estatico, container, false);

		Button boton = (Button) rootView.findViewById(R.id.button$fragment_estatico$cambiarFragmentsDinamicos);
		boton.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				mListener.onClickCambiarFragmentDinamico();
			}
		});

		return rootView;
	}

	@Override
	public void onAttach(Activity activity) {
		super.onAttach(activity);

		try {
			mListener = (OnMiClickListener) activity;
		} catch (ClassCastException e) {
			throw new IllegalStateException("La clase " + activity.toString() + " debe implementar de la interfaz " + OnMiClickListener.class.getName() + " del Fragment al que quiere escuchar");
		}
	}

	@Override
	public void onDetach() {
		super.onDetach();
		mListener = mListnerVacio;
	}

}
```

MainActivity.java

```java
public class MainActivity extends Activity implements OnMiClickListener {

	private boolean mPermutador = true;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		DinamicoFragmentA fragmentDinA = new DinamicoFragmentA();

		getFragmentManager().beginTransaction()
			.add(R.id.frameLayout$activity_main$contenedorFragmentsDinamicos, fragmentDinA)
			.commit();
	}

	@Override
	public void onClickCambiarFragmentDinamico() {
		Fragment fragmentElegido = null;

		if (mPermutador) {
			fragmentElegido = new DinamicoFragmentB();
		} else {
			fragmentElegido = new DinamicoFragmentA();
		}

		getFragmentManager().beginTransaction()
			.replace(R.id.frameLayout$activity_main$contenedorFragmentsDinamicos, fragmentElegido)
			.commit();

		mPermutador = !mPermutador;
	}
}
```

res/layout/activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:divider="?android:attr/dividerHorizontal"
    android:orientation="vertical"
    android:showDividers="middle"
    tools:context="com.jarroba.deunfragmentoaotro.MainActivity" >
```

```
    <fragment
        android:id="@+id/fragment$activity_main$fragmentEstatico"
        android:name="com.jarroba.deunfragmentoaotro.EstaticoFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="3"
        tools:layout="@layout/fragment_estatico" />

    <FrameLayout
        android:id="@+id/frameLayout$activity_main$contenedorFragmentsDinamicos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

res/layout/fragment_dinamico_a.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_purple" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Fragment Dinámico A"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

res/layout/fragment_dinamico_b.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_light" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Fragment Dinámico B"
        android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```

res/layout/fragment_estatico.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light"
    android:gravity="center"
    android:orientation="vertical" >

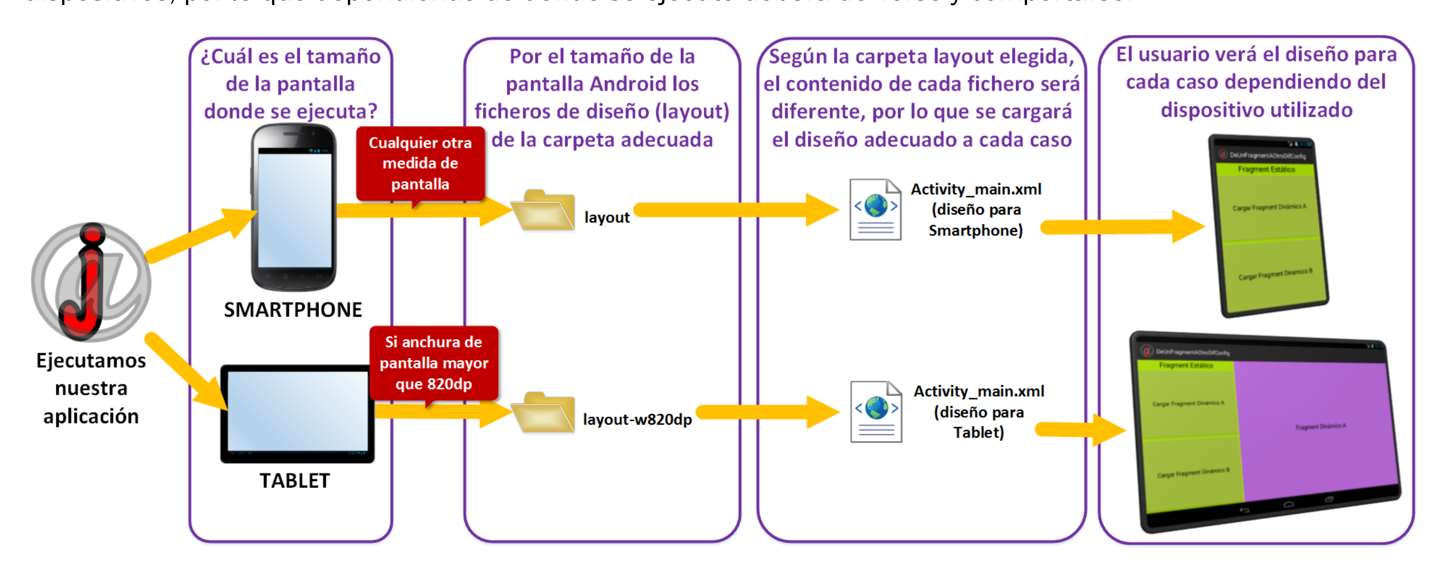
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment Estático"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$fragment_estatico$cambiarFragmentsDinamicos"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Cambiar de Fragment Dinámico" />
</LinearLayout>
```

Ejemplo de cambiar de un Fragment a otro en diferentes dispositivos

Lo que haremos

Queremos una aplicación con dos botones. Al pulsar cada botón cargará un contenido diferente de manera dinámica (un Fragment diferente para cada contenido). Además, será una aplicación para diferentes dispositivos, por lo que dependiendo de dónde se ejecute deberá de verse y comportarse:



- En tamaño Tablet o superior: aparecerán los botones a la izquierda y el Fragment con el contenido a la derecha. Además, se animará la entrada de cada Fragment con un leve difuminado. Como hicimos en el anterior ejemplo, al arrancar por primera vez la aplicación cargaremos un Fragment para que el usuario no tenga ni que molestarse en pulsar la primera vez ningún botón.
- En el resto de dispositivos más pequeños (las imágenes de aquí veremos un Smarthpone): En una Activity se mostrarán los botones. Debido al poco espacio en la pantalla, en otro Activity se mostrará el Fragment con el contenido elegido en cada uno de los botones.



Ilustración 26 – En Tablets, al iniciar la aplicación carga por defecto el Fragment A. Pulsamos para cargar el Fragment B.

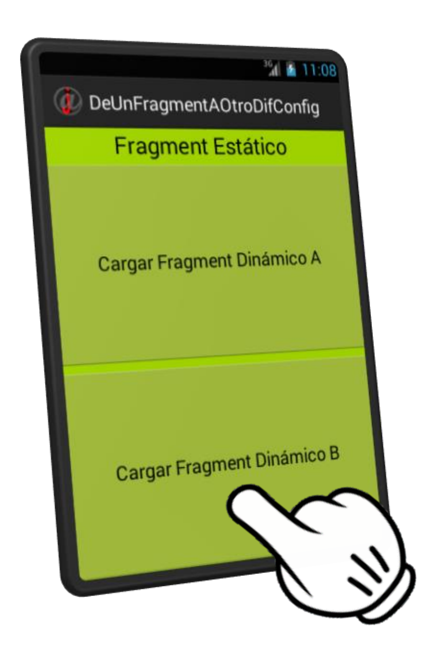
Se carga el Fragment B. Si pulsamos de nuevo en Fragment A, se eliminará el Fragment B y se volverá a cargar el Fragment A.



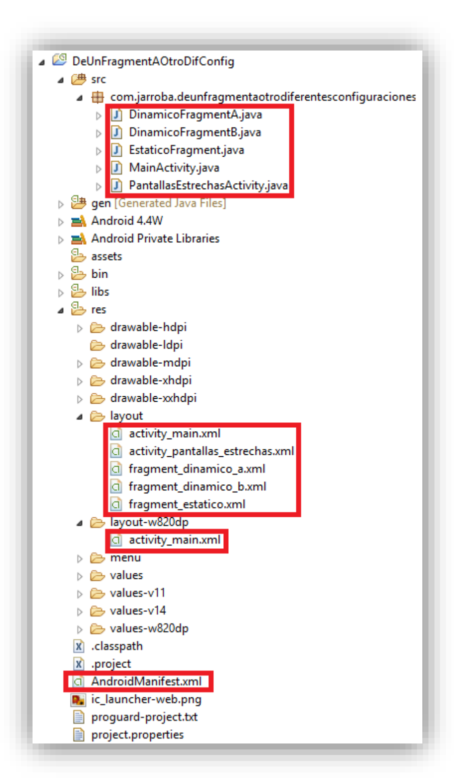
Ilustración 27 - Si ejecutamos la aplicación en un Smartphone, solo se cargará la Activity con los dos botones, por falta de espacio. Pulsando uno se cargará una nueva Activity con el Fragment correspondiente



Ilustración 28 - Una vez cargada pulsaremos el botón de Atrás del dispositivo. Y volveremos a los dos botnes donde podremos elegir el otro Fragment







Reutilizamos del anterior ejemplo:

- DinamicoFragmentA
- DinamicoFragmentB

EstaticoFragment.java

```java
public class EstaticoFragment extends Fragment {

        public final static int FLAG_FRAGMENT_A = 1;
        public final static int FLAG_FRAGMENT_B = 2;

        public interface OnMiClickListener {
                public void onClickCambiarFragmentDinamico(int flagFragment);
        }

        private static OnMiClickListener mListnerVacio = new OnMiClickListener() {
                @Override
                public void onClickCambiarFragmentDinamico(int flagFragment) {}
        };

        private OnMiClickListener mListener = mListnerVacio;

        public EstaticoFragment() {}

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
                View rootView = inflater.inflate(R.layout.fragment_estatico, container, false);

                Button botonA = (Button) rootView.findViewById(R.id.button$fragment_estatico$cargarFragmentDinamicoA);
                botonA.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                mListener.onClickCambiarFragmentDinamico(FLAG_FRAGMENT_A);
                        }
                });

                Button botonB = (Button) rootView.findViewById(R.id.button$fragment_estatico$cargarFragmentDinamicoB);
                botonB.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                mListener.onClickCambiarFragmentDinamico(FLAG_FRAGMENT_B);
                        }
                });

                return rootView;
        }
```

```
	@Override
	public void onAttach(Activity activity) {
		super.onAttach(activity);

		try {
			mListener = (OnMiClickListener) activity;
		} catch (ClassCastException e) {
			throw new IllegalStateException("La clase " + activity.toString() + " debe implementar de la
interfaz " + OnMiClickListener.class.getName() + " del Fragment al que quiere escuchar");
		}
	}

	@Override
	public void onDetach() {
		super.onDetach();
		mListener = mListnerVacio;
	}

}
```

MainActivity.java

```
public class MainActivity extends Activity implements OnMiClickListener {

	private boolean mDosFragmentos = false;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		View frameLayoutContenedor = findViewById(R.id.frameLayout$activity_main$contenedorFragmentsDinamicos);

		mDosFragmentos = frameLayoutContenedor != null && frameLayoutContenedor.getVisibility() == View.VISIBLE;

		if (mDosFragmentos) {
			this.onClickCambiarFragmentDinamico(EstaticoFragment.FLAG_FRAGMENT_A);
		}

	}

	@Override
	public void onClickCambiarFragmentDinamico(int flagFragment) {

		if (mDosFragmentos) {
			Fragment fragmentElegido = null;
			switch (flagFragment) {
				case EstaticoFragment.FLAG_FRAGMENT_A: {
					fragmentElegido = new DinamicoFragmentA();
					break;
				}
				case EstaticoFragment.FLAG_FRAGMENT_B: {
					fragmentElegido = new DinamicoFragmentB();
					break;
				}
			}

			getFragmentManager().beginTransaction()
				.replace(R.id.frameLayout$activity_main$contenedorFragmentsDinamicos, fragmentElegido)
				.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
				.commit();
		} else {
			Intent intent = new Intent();
			intent.setClass(this, PantallasEstrechasActivity.class);
			intent.putExtra(PantallasEstrechasActivity.KEY_FLAG_FRAGMENT_ENVIADO, flagFragment);
			startActivity(intent);
		}
	}

}
```

PantallasEstrechasActivity.java

```
public class PantallasEstrechasActivity extends Activity {

	public final static String KEY_FLAG_FRAGMENT_ENVIADO = "Clave atributo que recibe este fragment";

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_pantallas_estrechas);

		if (savedInstanceState == null) {
			Bundle extras = getIntent().getExtras();
			int flagFragment = extras.getInt(KEY_FLAG_FRAGMENT_ENVIADO);

			Fragment fragmentElegido = null;
			switch (flagFragment) {
				case EstaticoFragment.FLAG_FRAGMENT_A: {
					fragmentElegido = new DinamicoFragmentA();
					break;
				}
				case EstaticoFragment.FLAG_FRAGMENT_B: {
					fragmentElegido = new DinamicoFragmentB();
					break;
				}
			}

			getFragmentManager().beginTransaction()
				.add(R.id.frameLayout$activity_pantallas_estrechas$contenedorFragmentsDinamicos,
fragmentElegido)
				.commit();
		}
	}

}
```

res/layout/activity_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/FrameLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.jarroba.deunfragmentaotrodiferentesconfiguraciones.PantallasEstrechasActivity" >

    <fragment
        android:id="@+id/fragment$activity_main$fragmentEstatico"
        android:name="com.jarroba.deunfragmentaotrodiferentesconfiguraciones.EstaticoFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout="@layout/fragment_dinamico_a" />

</FrameLayout>
```

res/layout/activity_pantallas_estrechas.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frameLayout$activity_pantallas_estrechas$contenedorFragmentsDinamicos"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</FrameLayout>
```

Reutilizamos del anterior ejemplo el código de:

- res/layout/fragment_dinamico_a.xml
- res/layout/fragment_dinamico_b.xml

res/layout/fragment_estatico.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_green_light"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
```

```xml
        android:layout_height="wrap_content"
        android:text="Fragment Estático"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$fragment_estatico$cargarFragmentDinamicoA"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Cargar Fragment Dinámico A" />

    <Button
        android:id="@+id/button$fragment_estatico$cargarFragmentDinamicoB"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Cargar Fragment Dinámico B" />

</LinearLayout>
```

res/layout-w820dp/activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:divider="?android:attr/dividerHorizontal"
    android:orientation="horizontal"
    android:showDividers="middle"
    tools:context="com.jarroba.deunfragmentaotrodiferentesconfiguraciones.MainActivity" >

    <fragment
        android:id="@+id/fragment$activity_main$fragmentEstatico"
        android:name="com.jarroba.deunfragmentaotrodiferentesconfiguraciones.EstaticoFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="2"
        tools:layout="@layout/fragment_estatico" />

    <FrameLayout
        android:id="@+id/frameLayout$activity_main$contenedorFragmentsDinamicos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.deunfragmentaotrodiferentesconfiguraciones"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name="PantallasEstrechasActivity"></activity>

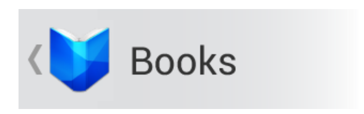
    </application>
</manifest>
```

Navegación hacia atrás y hacia arriba

¿Navegar hacia atrás y hacia arriba?

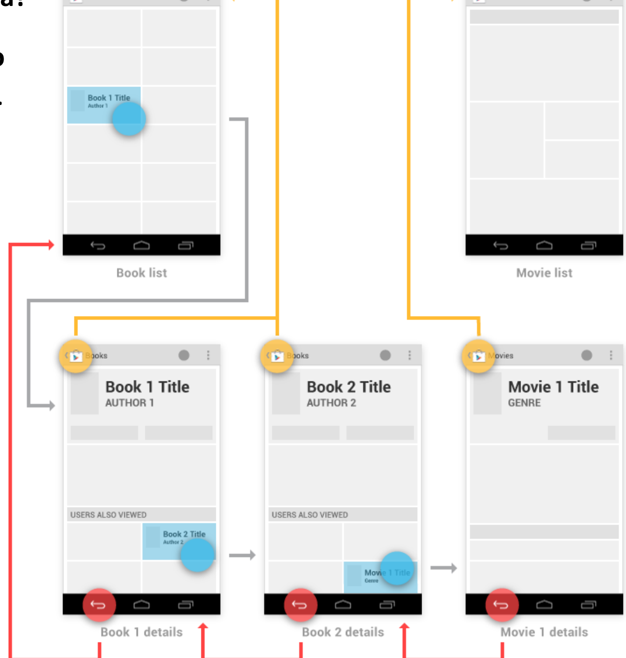


Si eres usuario de algún dispositivo con Android, tendrás en la cabeza como **navegar hacia atrás**: pulsando el botón de atrás. Botón que antiguamente solía ser físico y ahora siempre se encuentra en la parte inferior de la pantalla sobre fondo negro, indicado el botón con una flecha de “atrás”.



Sin embargo, **navegar hacia arriba** te puede sonar un poco extraño. El botón de navegar hacia arriba se ubica en la parte superior de la pantalla, en el interior del ActionBar (barra de acciones), a la izquierda con forma de “<” junto al icono (y a veces el nombre de la aplicación).

¿Qué diferencia hay entre pulsar el botón de atrás y el de arriba?

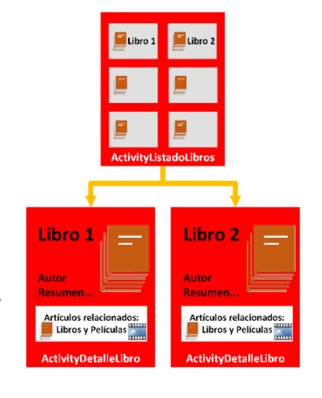


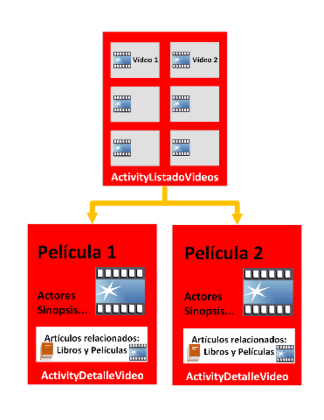
- **Back Button** (Botón de atrás): Navega en orden **opuesto al cronológico** tanto dentro como fuera de la aplicación. También soporta cerrar ventanas de dialogo o popups, eliminar elementos seleccionados, cerrar el ActionBar contextual, u ocultar el teclado. Des-apila Activities de la pila de Activity. Puede hacer salir de la aplicación.
- **Up Button** (Botón de arriba): Navega **en base a la jerarquía de las relaciones entre Activities** dentro de la aplicación. Apila Activities en la pila de Activity. Nunca hace sale de la aplicación.

Veamos un ejemplo con la aplicación “Google Play” donde compramos y descargamos otras aplicaciones.

Empezamos en el listado de libros y elegimos un libro. Nos abrirá la ficha del libro donde podremos hacer 3 cosas:

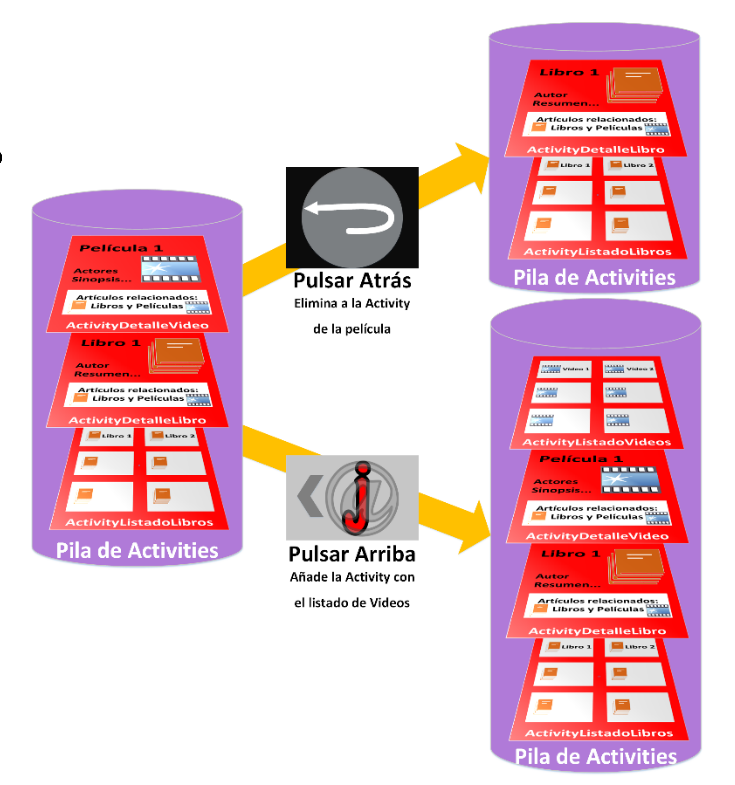
1. Pulsar “atrás”: volverá al listado de libros
2. Pulsar “arriba”: volverá al listado de libros
3. Pulsar en un artículo sugerido. Podremos elegir entre:
    a. Un libro. En la ficha de libro podremos:
        i. Pulsar “atrás”: volverá al libro anterior
        ii. Pulsar “arriba”: volverá al listado de libros
        iii. Pulsar en un artículo sugerido (podremos seguir abriendo ramas, para simplificar lee el siguiente punto).
    b. Una película. En la ficha de película podremos:
        i. Pulsar “atrás”: volverá al libro anterior
        ii. Pulsar “arriba”: volverá al listado de películas
        iii. Pulsar en un artículo sugerido (y así infinitamente)





Esto es así, ya que la jerarquía de Activies para este caso es la de la imagen de la derecha. Donde tenemos dos árboles, uno para los libros y otro para las películas. En cada listado al pulsar un elemento se abrirá el detalle del elemento apropiado.

De este modo, si nos encontramos en el punto del ejemplo donde: desde el listado de libros hemos escogido un libro, y luego desde los artículos relacionados hemos elegido una película. Tendremos una pila de Activity como la que se muestra a la derecha de la imagen de al lado (cada Activity ya pasada es la que está más abajo, la que ve el usuario es la que está en la cima de la pila).



Aquí si pulsamos el botón de atrás, la Activity con el detalle de la película se des-apilará.

Pero si pulsamos el botón de arriba, siguiendo la jerarquía como vimos antes, se apilará la Activity del listado de películas.

Por lo que dependiendo lo que pulsemos tendremos de resultado una pila de Activities u otra.

Tendremos que tener cuidado a qué Activity volverá el usuarios cuando pulse un botón u otro.

¿Cómo puedo programar el botón de arriba?

Es bastante sencillo.

Primero tenemos que declarar la estructura jerárquica. Esto se hace desde el AndroidManifest.xml utilizando para cada hijo el atributo “android:parentActivityName”. Con esto conseguiremos dos cosas: desde la Activity en la que estemos saber cuál es su padre para ir al adecuado y que Android nos añada automáticamente el botón de arriba sobre el ActionBar.

Por ejemplo, si queremos diseñar la siguiente jerarquía de la derecha:

Libros

LibroA LibroB

Pondremos en nuestro AndroidManifest.xml la siguiente estructura:

```
<activity
        android:name=".Libros" >
</activity>

<activity
        android:name=".LibroA"
        android:parentActivityName=".Libros" >
</activity>

<activity
        android:name=".LibroB"
        android:parentActivityName=".Libros" >
</activity>
```

Ahora solo nos queda en Activity escuchar el evento de clicado sobre el botón de arriba, y con este abrir al padre adecuado. Para abrir al padre nos basta con tener un Intent ya formado con la clase del padre a abrir (Imaginemos que estamos en “LibroA” por ejemplo), por lo que es tan sencillo como llamar desde la Activity en la que estemos al método:

```
Intent upIntent = getParentActivityIntent();
```

Ya tendremos al Intent con la clase del Activity padre que queremos abrir (en este ejemplo un Intent con “Libros”).

Ya solo nos queda llamar a “startActivity()” y pasarle el Intent, pero también tenemos que decirle que nos limpie la pila de atrás para no tener un sinfín de Activities guardadas (Es decir, con el ejemplo si estamos en “Libros” vamos a “LibroA” luego pulsamos el botón de arriba nos volvería a crear otra Activity “Libros” y no queremos

esto; sino que solo tengamos un único Activity "Libros"). Para hacer esto existen varios modos, aquí pondremos el siguiente ejemplo, donde añadiremos un flag al Intent que haga justo esto anterior, des-apile de la pila de atrás todas las Activities hasta que tengamos solo una de ese tipo. Se puede hacer como:

```
upIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(upIntent);
finish();
```

Por fortuna Android nos facilita la vida con un método simple que está en Activity para hacer lo anterior, por lo que podemos utilizar (Nota: por la información existente y por las pruebas, este método parece tener un bug o un cambio de comportamiento no reportado, no cumple con la teoría del botón de arriba en todos los casos):

```
navigateUpTo(upIntent);
```

Ahora solo queda introducir todo este código en el evento que escuche. Este evento es el correspondiente con el menú de opciones, sobrescribiendo el método "onOptionsItemSelected()". Dentro de este método nos llega un MenuIntent que nos dirá el elemento que se ha seleccionado sobre la barra de tareas. Para escuchar solo al botón de atrás será el MenuItem coincidente con el id "android.R.id.home". Un switch nos facilitará mucho:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()) {
                case android.R.id.home:

                        Intent upIntent = getParentActivityIntent();
                        navigateUpTo(upIntent);

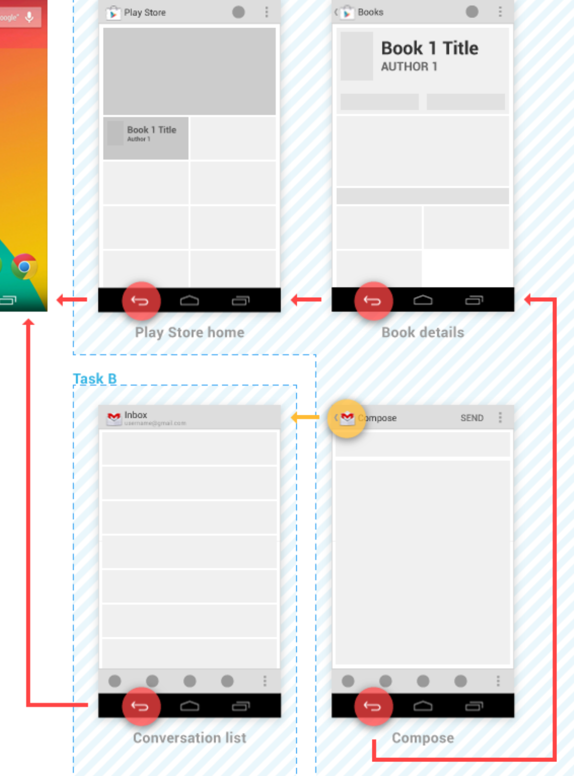
                        return true;
                }

        return super.onOptionsItemSelected(item);
}
```

¿Existen casos donde el botón de arriba se comporta igual que el de atrás?

Sí, en los casos donde coincida el orden cronológico y el de jerarquía.

¿Puedo ir a otra aplicación al pulsar el botón de arriba? ¿Qué es una tarea?



Hay unos casos especiales donde pulsando el botón de arriba nos moveríamos a otra aplicación. Entonces puedes preguntar: ¿No hemos dicho que el botón de arriba no puede salir de la aplicación? Es un caso peculiar, pues no salimos de la jerarquía por lo tanto no salimos de la aplicación pero si cambiamos de tarea (task).

Aquí entra el concepto de **Tarea (Task)**: **Secuencia de Activities que un usuario sigue para conseguir su objetivo**. Pueden ser tanto Activities de la misma App o de otras Apps de terceros.

Explico con un ejemplo. Supongamos que abrimos una aplicación cualquiera (como "Google play"). Esta aplicación se abrirá sobre una tarea (Task A). Navegamos por la aplicación y en cierto momento pulsamos el botón de compartir. Aquí elegimos que queremos compartir por correo electrónico (como "Gmail"). Esto nos abrirá una Activity para redactar el correo de la aplicación de correo, que estará en la misma tarea que la primera aplicación -esto es así porque el usuario quiere seguir usando la primera aplicación y siente

como que la Activity de redactar el correo está dentro de la misma primera aplicación. Si pulsamos en enviar el correo se enviará y cerrará la Activity del correo; por lo que será lo mismo que pulsar atrás, volveremos a la Activitiy anterior. Sin embargo, si pulsamos el botón de arriba, por jerarquía de Activities llegaremos al listado de correos -a la bandeja de entrada del correo- y a partir de aquí ya estaremos en otra tarea (Task B) diferente -ya que el deseo del usuario ya no es estar en la primera aplicación, sino en otra diferente que es la de correo. De este modo la primera tarea se pone en segundo plano y la nueva tarea pasará a estar en primer plano.

Si otras aplicaciones pueden usar una de mis Activities ¿Cómo puedo diferenciar si estoy en la tarea de mi aplicación o en otra de terceros?

En caso de que nuestra Activity se pueda abrir desde una aplicación externa, será importante diferenciar si estamos en la misma tarea o en otra. Para ello utilizaremos el método shouldUpRecreateTask() que nos dirá si estamos en otra tarea si es "true", si "false" estaremos en nuestra aplicación. Luego solo queda sincronizar la pila de atrás con TaskStackBuilder añadiendo la Activity a la pila de atrás e indicar que queremos navegar hacia arriba.

```
if (shouldUpRecreateTask(upIntent)) {
        TaskStackBuilder.create(this)
                .addNextIntentWithParentStack(upIntent)
                .startActivities();
} else {
        navigateUpTo(upIntent);
}
```

¿Es siempre necesario poner el botón de arriba en todas las aplicaciones?

Siempre que haga falta, ya que una aplicación no tiene un lugar principal por donde iniciarse, deberemos guiar al usuario desde donde esté. Por ejemplo como vimos antes, si desde otra aplicación pulsamos en compartir y redactar correo, se nos abrirá la aplicación de correo pero desde la Activity de redacción, no desde la del listado de correos; para llegar a esta última pulsaremos el botón de arriba.

¿Modificar una View en una Activity, modifica el comportamiento de los botones arriba o atrás?

No cambia el comportamiento de los botones Atrás o Arriba. Por ejemplo: hacer switch entre tabs, filtrar una lista, usar zoom, etc.

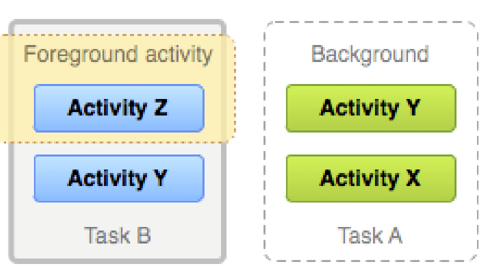
¿Cuál es el comportamiento con varios puntos de entrada a la aplicación?

Algunas Activities no tienen un camino fijo por el que llegar, pudiendo acceder desde diferentes lugares. El botón de arriba, en este caso se comportaría igual que el de atrás. Ejemplo: una pantalla de opciones.

¿Y el comportamiento del botón de Home?



Pulsar el Home Button (Botón de salida al Escritorio) provoca la salida completa al "HomeScreenLauncher".



La tarea completa (en la imagen de ejemplo Task A), por tanto, también su pila de atrás, se ponen en segundo plano, por lo que pierde el foco. Entonces el usuario puede arrancar una nueva tarea (en la imagen Task B) que se pondrá en primer plano. La tarea queda en segundo plano, pero su pila de atrás guarda el estado. Así, de volver a ser seleccionada por el usuario, recuperará al estado en que quedó previamente. Cabe notar que el sistema puede destruir las

tareas en segundo plano si necesitara los recursos. Por si fuera destruida la Activity, es importante salvar su estado con onSaveInstanceState()

¿Puedo arrancar una Activity varias veces?

Es posible crear nuevas instancias de la misma Activity varias veces (desde la misma tarea u otra diferente), por lo que en la pila de atrás se comportarían como si fueran Activities diferentes.

Activity 1
Home Activity
Activity 2
Home Activity
Back Stack
Separate instances

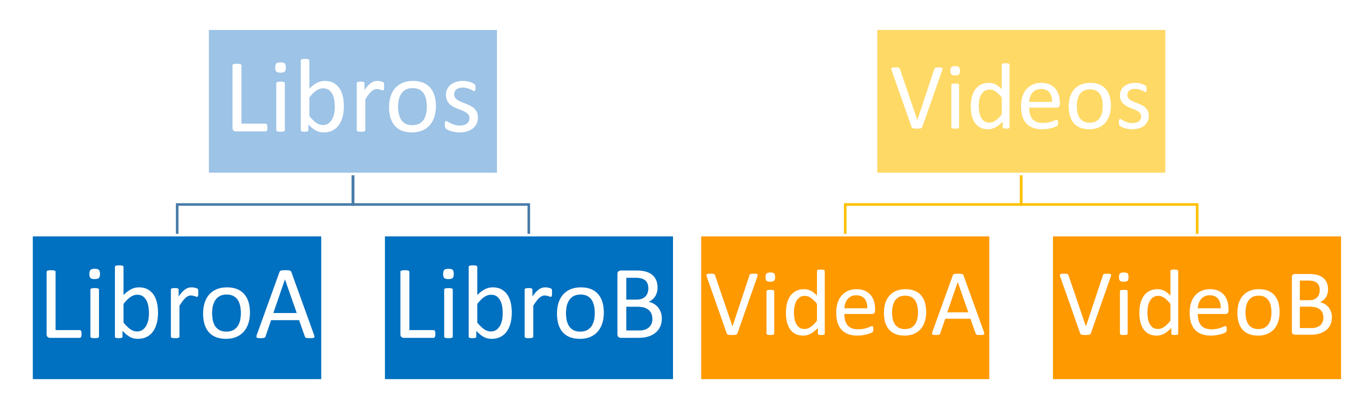
Referencias:

- http://developer.android.com/guide/topics/ui/actionbar.html
- http://developer.android.com/design/patterns/navigation.html
- http://developer.android.com/training/implementing-navigation/ancestral.html
- http://developer.android.com/reference/android/support/v4/app/NavUtils.html
- http://developer.android.com/training/design-navigation/ancestral-temporal.html

Ejemplo de navegar hacia atrás y hacia arriba

Lo que haremos

Simularemos que hacemos la aplicación de Google Play indicada anteriormente. Queremos que exista un listado de libros y otro de vídeos. Podríamos poner un menú de selección de listados, por simplificar, iniciaremos la aplicación sobre el listado de libros directamente. También, por simplificar, haremos los dos listados con botones (más adelante haremos los listados con ListView que es como hay que hacerlos en Android). Evidentemente, al pulsar sobre un elemento del listado se nos abrirá el detalle de dicho elemento (si es un libro su descripción, título, autor, etc; y si es un vídeo su sinopsis, actores, etc). Otra vez simplificaremos poniendo los datos del detalle a pelo en una Activity aparte (cuando veamos listados veremos que añadir el detalle será dinámico). De este modo nos quedará la siguiente estructura de la aplicación (es igual que la que vimos previamente en la pregunta "¿Qué diferencia hay entre pulsar el botón de atrás y el de arriba?"):



Este ejemplo está diseñado para que te hagas a la idea de la estructura de listado, tampoco queremos meternos en profundidad con los listados todavía, así que simplificaremos más haciendo solo el LibroA y el VideoA para el ejemplo.

No tenemos que olvidar colocar a cada libro y a cada vídeo el botón de arriba.

También añadiremos a cada libro y cada vídeo una parte de productos recomendados (lo haremos en un Fragment estático para poder reutilizarlo en varios sitios)

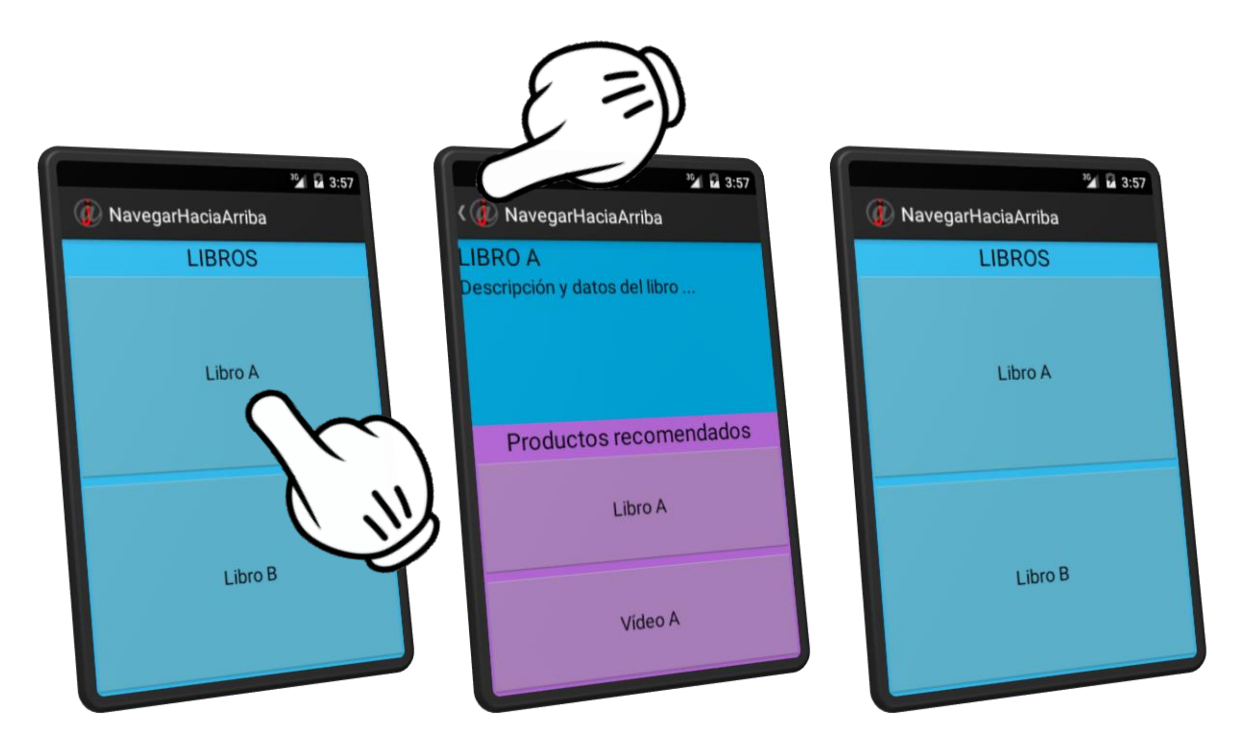


Ilustración 29 - Probaremos primero dirigiéndonos a un detalle de un libro y pulsaremos el botón de arriba. Hará lo mismo que pulsar el botón de atrás, con la diferencia interna que ha vuelto por el nivel de jerarquía no de historial



Ilustración 30 – Otro ejemplo de pulsar arriba es si ahora estando en el detalle del libro pulsamos en un vídeo recomendado. En el detalle del vídeo pulsaremos el botón de arriba, en este caso sí que el usuario ha notado la diferencia, hemos vuelto por la jerarquía hasta el listado de vídeos, no el de libros

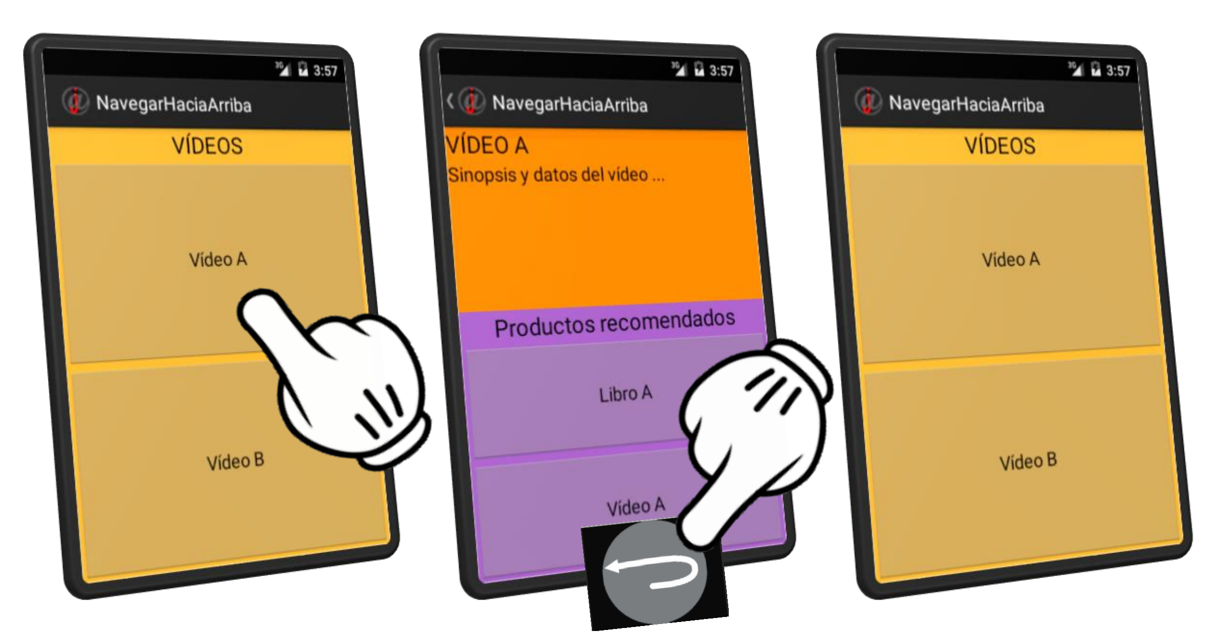
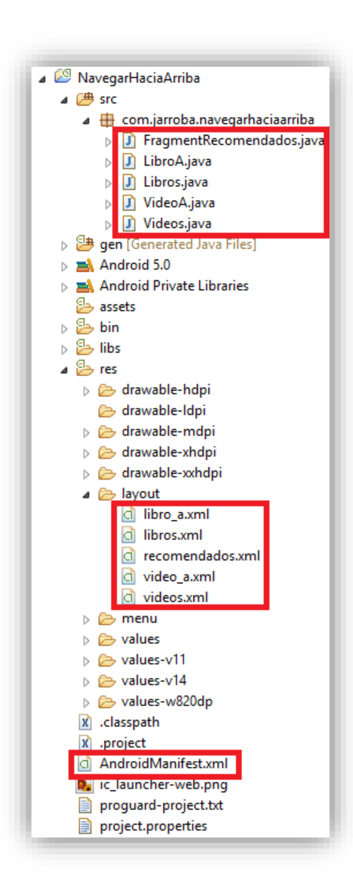


Ilustración 31 - Solo nos queda el caso del botón de atrás. Si estamos en cualquier detalle y pulsamos atrás volveremos por el historial. Como podemos comprobar, hace lo mismo que el primer caso aquí mostrado, pero des-apilando una Activity de la pila de atrás, no volviendo por la jerarquía



FragmentRecomendados.java

```
public class FragmentRecomendados extends Fragment {

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
		View rootView = inflater.inflate(R.layout.recomendados, container, false);

		Button bLibroA = (Button) rootView.findViewById(R.id.button$libros$libroA);
		bLibroA.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				Intent intent = new Intent(getActivity(), LibroA.class);
				startActivity(intent);
			}
		});

		Button bVideoA = (Button) rootView.findViewById(R.id.button$videos$videoA);
		bVideoA.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				Intent intent = new Intent(getActivity(), VideoA.class);
				startActivity(intent);
			}
		});

		return rootView;
	}

}
```

LibroA.java

```
public class LibroA extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.libro_a);
	}
```

```java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case android.R.id.home:

            Intent upIntent = getParentActivityIntent()
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(upIntent);
            finish();

            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Libros.java

```java
public class Libros extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.libros);

        Button bIrLibroA = (Button) findViewById(R.id.button$libros$libro1);
        bIrLibroA.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(), LibroA.class);
                startActivity(intent);
            }
        });

    }

}
```

VideoA.java

```java
public class VideoA extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.video_a);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case android.R.id.home:

            Intent upIntent = getParentActivityIntent()
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(upIntent);
            finish();

            return true;
        }

        return super.onOptionsItemSelected(item);
    }

}
```

Videos.java

```java
public class Videos extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.videos);

		Button bIrVideoA = (Button) findViewById(R.id.button$videos$videoA);
		bIrVideoA.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				Intent intent = new Intent(getApplicationContext(), VideoA.class);
				startActivity(intent);
			}
		});

	}

}
```

res/layout/libro_a.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_dark"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.navegarhaciaarriba.libro_a" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/libro_a"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/descripci_n_y_datos_del_libro_"
            android:textAppearance="?android:attr/textAppearanceMedium" />
    </LinearLayout>

    <fragment
        android:id="@+id/fragment$libro_a$fragmentRecomendados"
        android:name="com.jarroba.navegarhaciaarriba.FragmentRecomendados"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        tools:layout="@layout/recomendados" />

</LinearLayout>
```

res/layout/libros.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_light"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.navegarhaciaarriba.Libros" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="LIBROS"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$libros$libro1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Libro A" />

    <Button
        android:id="@+id/button$libros$libro2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Libro B" />

</LinearLayout>
```

res/layout/recomendados.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_purple"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.navegarhaciaarriba.FragmentRecomendados" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Productos recomendados"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$libros$libroA"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Libro A" />

    <Button
        android:id="@+id/button$videos$videoA"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Vídeo A" />

</LinearLayout>
```

res/layout/video_a.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_orange_dark"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.navegarhaciaarriba.video_a" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/v_deo_a"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/sinopsis_y_datos_del_v_deo_"
            android:textAppearance="?android:attr/textAppearanceMedium" />
    </LinearLayout>

    <fragment
        android:id="@+id/fragment$libro_b$fragmentRecomendados"
        android:name="com.jarroba.navegarhaciaarriba.FragmentRecomendados"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        tools:layout="@layout/recomendados" />

</LinearLayout>
```

res/layout/videos.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_orange_light"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.jarroba.navegarhaciaarriba.videos" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="VÍDEOS"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button$videos$videoA"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Vídeo A" />

    <Button
        android:id="@+id/button$videos$videoB"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Vídeo B" />

</LinearLayout>
```

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.navegarhaciaarriba"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".Libros"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".LibroA"
            android:parentActivityName=".Libros" >
        </activity>

        <activity
            android:name=".Videos" >
        </activity>

        <activity
            android:name=".VideoA"
            android:parentActivityName=".Videos" >
        </activity>

    </application>

</manifest>
```

Manifest

¿Qué es?

El Manifest representa la información esencial acerca de la App. Información que Android debe conocer antes de ejecutar el código de la App.

Android es quien se encarga de instanciar los componentes básicos de la aplicación (Activity, Services, Broadcast Receiver y Content Provider). Quiere decir, que si por ejemplo tenemos una Activity como la siguiente:

```
public class MiActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.mi_layout_de_mi_activity);
        }
}
```

Android la instanciará en algún sitio como lo solemos hacer en Java:

```
MiActivity ma = new MiActivity();
```

Solo que nosotros nunca escribiremos la anterior línea. Lo hará Android cuando tenga que hacerlo y con ello gestionará el ciclo de vida de la Activity.

Para declarar la anterior clase en el AndroidManifest.xml, tanta solo tendremos que añadir dentro de la etiqueta "<application>" la Activity a declarar. Para el ejemplo bastará con añadir la línea:

```
<activity android:name="miActivity"></activity>
```

Por lo que nuestro AndroidManifest.xml podría quedar tal que así (se explicará cada parte):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.miAplicacion"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".ActivityMain"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name="miActivity"></activity>
    </application>

</manifest>
```

¿Qué es debo declarar?

Algunas cosas que deben ser declaradas (los trozos de código de ejemplo son del ejemplo anterior):

- El **paquete Java** que servirá como único identificador de la aplicación, y la **versión** de nuestra aplicación

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.miAplicacion"
    android:versionCode="1"
    android:versionName="1.0" >
```

- Descripción de los **componentes de la aplicación:** Activity, Services, BroadcastReceiver y ContentProvider

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".ActivityMain"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name="miActivity"></activity>
</application>
```

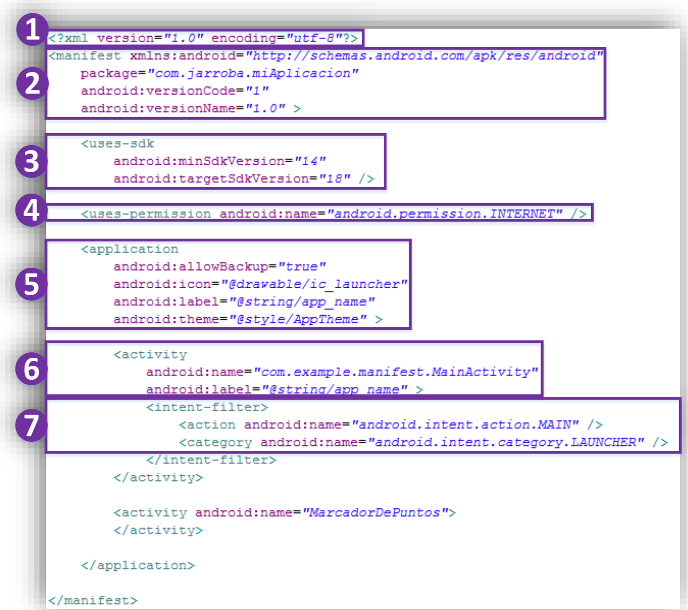
- Declarar los **permisos** que la App requiere para acceder a partes protegidas de la API o para interaccionar con otras Apps (en el ejemplo anterior no se utilizan)

```
<uses-permission android:name="android.permission.INTERNET"/>
```

- El **nivel de la API mínimo** y para qué **versión se compilará** la App

```
<uses-sdk
    android:minSdkVersion="16"
    android:targetSdkVersion="21" />
```

¿Cuáles son las partes sus básicas?



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jarroba.miAplicacion"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="18" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.manifest.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name="MarcadorDePuntos">
        </activity>

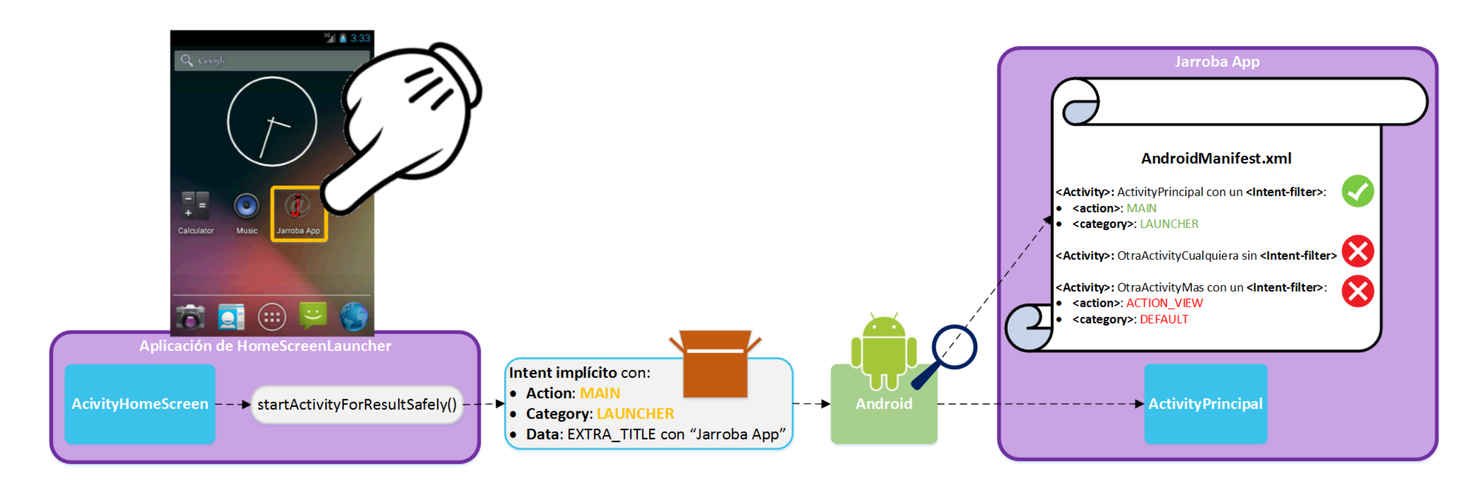
    </application>

</manifest>
```

1. **<?xml>:** Declaración de la codificación para representar los caracteres del documento
2. **<manifest>:** Raíz de AndroidManifest.xml que contiene el espacio de nombres, el paquete, la versión del código interna (para Google Play es obligatorio su incremento cada vez que se actualiza la aplicación), y la versión que verá el usuario
3. **<uses-sdk>:** La compatibilidad de la App. Contiene la versión mínima de Android que soportará la App, y que versión de comportamientos y apariencias tendrá la App
4. **<uses-permission>:** Solicitud de permisos de la aplicación al sistema para poder hacer uso de determinadas funciones. Como: acceso a la tarjeta de memoria, a Internet, al GPS, etc
5. **<applicaiton>:** Declaración de los componentes de la App. Contiene la copia de seguridad automática, el icono, el nombre a la App y el tema
6. **<activity>:** Componente Actividad declarado. Tiene un nombre de la clase, y lo que aparecerá de nombre la parte superior de la pantalla
7. **<intent-filter>:** Especifica el tipo de intención que está permitido en la App.
    - **<action>:** Acción a realizar o quien la llevó a cabo. Ha de empieza por: android.intent.action.ACCION
    - **<category>:** Nombre de la categoría con información adicional acerca del componente que deberá manejar el intent. Empiezan siempre con: android.intent.category.CATEGORIA

¿Qué significa la etiqueta intent-filter que siempre tiene la Activity que arranca Android?

Como su propio nombre indica es un filtro de intenciones. Cuando declaramos un Intent implícito (intención de hacer algo en alguna aplicación que nuestra aplicación no conoce) y lo lanzamos, Android se encarga de buscar entre todos los AndroidManifest.xml de todas las aplicaciones instaladas en el dispositivo. Android comprueba cuales <intent-filter> permiten que el Intent lanzado pase; lo hará mediante las etiquetas <action> y <category>, donde Android mirará si el “category” y “action” del Intent coinciden con las etiquetas <action> y <category> del <intent-filter>.



Referencias:

- http://developer.android.com/guide/topics/manifest/manifest-intro.html

Continuará

El libro de “Android 100%” es un libro en continua evolución y mejora. Así como lo es el mundo Android.

Iremos publicando material adicional a medida que lo tengamos. Se puede decir que es un libro sin fin.

No te pierdas ninguna novedad en www.Jarroba.com



Agradecimientos

Como saben todos los que conozco, me han ayudado, me han instruido, dado ánimos, hasta insistido en que siga hacia adelante, nunca me han faltado las palabras de agradecimiento. Tantos han sido los que me han influenciado que seguro me dejo a muchos, quiero que sepan que también los incluyo dentro de mí.

Primero quiero agradecer la posibilidad de escribir este libro a Dios. No hubiera conseguido escribir ni una sola línea sin disponer de los medios, de estar con la gente que me ha rodeado, sin estar en el contexto y en el país en el que vivo, y la confianza de que Él siempre está ahí para velar por nosotros. Por siempre gracias.

A mi familia Charo, Carlos y Guillermo. Su apoyo incondicional en todo cuanto he hecho siempre me ha llegado al corazón, son mi fortaleza y lo más importante para mí.

A Ricardo Moya García mi socio y cofundador de www.Jarroba.com. Siempre tirando con fuerza para que saliéramos adelante, aprendiendo juntos, marcando siempre la diferencia. Su fuerza interior por descubrir cosas nuevas, su gusto por el trabajo bien hecho y de estar a la última en todo ámbito informático.

A Jesús Alberto Casero mi otro socio y cofundador de www.inactec.com. Su increíble habilidad de adaptarse a los cambios, la facilidad con la que se desenvuelve y su capacidad por mantener la visión de equipo. No hubiéramos podido hacer muchas cosas sin él.

A mis amigos informáticos que siempre han estado cerca Julián Sánchez, Javier Eduardo Domínguez, Roberto Sáez, Víctor Suarez, Wiktor Nykiel, Alfonso Ayuso y a todos los demás. Su actitud siempre alegre y de superación personal, ha logrado que consigamos grandes cosas, que andemos por el buen camino.

Cómo olvidar a Francisco Javier Sánchez, Sebastián Naranjo, Luís de Frutos, entre otros muchos amigos que no tienen que ver nada con la informática. Han estado ahí para compartir nuevas experiencias, enseñarme sobre otros campos y mostrarme una perspectiva muy diferente.

A mis compañeros de trabajo, jefes, y a todos quien me han acompañado en este apasionante mundo de tecnología profesional, quien me ha dado la oportunidad de desarrollarme. En especial agradecimiento a César Ruíz, un gran líder muy organizado, le agradezco que me guiara y que me ensañara un sinfín de aplicativos.

A mis jefes también. Me han dado la oportunidad de mejorar tanto profesionalmente como individualmente.

A mis profesores de la universidad, con los que empecé a formar mi cabeza y quienes fueron fuente de inspiración. Frank Díaz, Juan Alberto de Frutos, Francisco Serradilla, Agustín Yagüe, Eugenio Santos, me dejo muchos en el tintero que también me marcaron en un antes y un después.

No puedo dejar de mencionar algunas web de habla hispana, que se esfuerzan por llevar una calidad que todavía no tenemos los informáticos -y la tecnología sea dicha- en el mundo hispano. No me pagan, no es por hacer publicidad, no tenemos ningún contrato ni tipo de afiliación, y tampoco me conocen -a lo sumo nos hemos cruzado cuando asistí a alguno de sus eventos como uno más del público. Simplemente me parecía justo mencionar a aquellas web que me han marcado de un modo u otro. Quiero mencionarlas para inspiración de los demás. Estas son: http://www.adictosaltrabajo.com/, http://www.javiergarzas.com/, http://www.elladodelmal.com/, https://mejorando.la/, http://www.sgoliver.net/, http://www.enriquedans.com/

A todos gracias.

Legales y derechos

El libro de “Android 100%” estará siempre disponible para su descarga desde el dominio de www.Jarroba.com. De obtenerse desde cualquier otra fuente, tanto el autor como sus ayudantes no darán ningún tipo de soporte, ni se hacen responsables por la posible inclusión de virus, retraso en el aprendizaje, material explícito, engaños, material anticuado, atentados contra derechos, etc.

Este libro contiene imágenes extraídas de http://developer.android.com/index.html que pertenecen a sus autores (se indica en los términos que los autores aparecen al lado de las imágenes, no se encontraron posibles atribuciones, no pudiendo citar a personas concretas), así como algunas traducciones realizadas por el autor de este libro intentando ser lo más fiel a las redacciones de http://developer.android.com. El autor estima que ha cumplido lo que específica las licencias “Apache 2.0 License” y “Creative Commons 2.5 Attribution License”; expuestos en http://developer.android.com/license.html; de no ser así, pónganse en contacto con el autor de este libro a través de correo jarrobaweb@gmail.com para indicar las correcciones pertinentes.

La fuente de la letra de este libro es “Franklin Gothic Book” que se encuentra en el paquete de Microsoft Office.

En el apartado de “agradecimientos” he citado a cada uno desde el mejor de mis deseos. Decir que si me he equivocado en algo, o no desearan aparecer, me lo hagáis saber para poder corregirlo cuanto antes. Del mismo apartado de “agradecimientos” hago notar que en ningún caso se ha copiado nada de lo que esas webs ofrecen, ni en este libro ni en www.Jarroba.com; nos hemos comprometido a crear material único, a explicar el material técnico de una manera muy diferente, ofrecido de otra manera, pero nunca pisando o robando el trabajo a nadie.

Todas las novedades y últimas versiones en exclusiva desde el dominio de www.jarroba.com



Este **libro está registrado oficialmente** desde:

Android 100%

CC by-nc-sa *Ramón Invarato Menéndez*

https://www.safecreative.org/work/1410272409903